

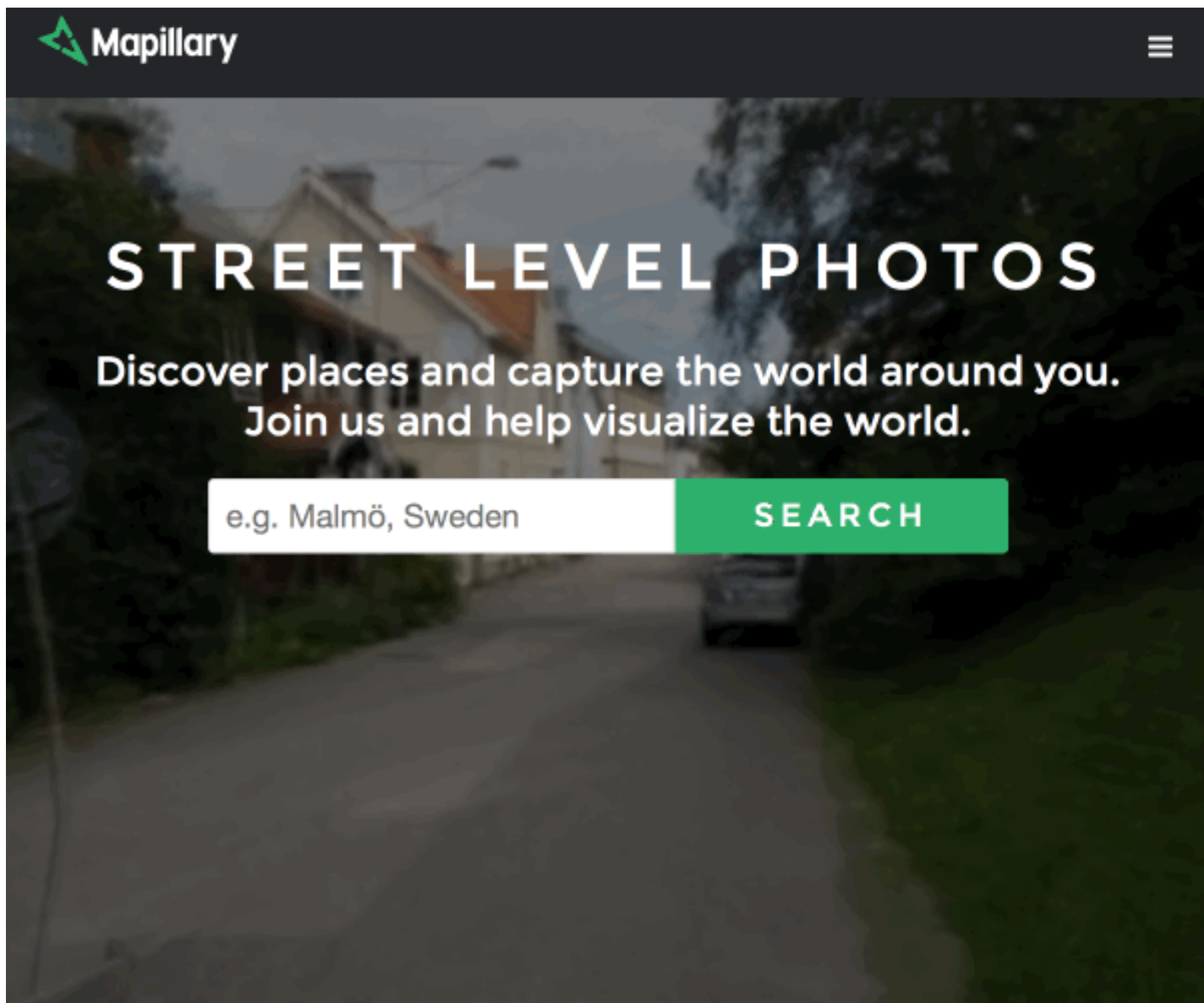
Some of Our Favorite Mapzen Search Projects of 2015

search (/tag/search)

It's been an exciting year for **Mapzen Search** (<https://mapzen.com/projects/search>) and for our users, so we thought today would be a great time to look back at some of the most exciting things you're doing with Mapzen Search!

Mapillary

Our friends at **Mapillary** (<https://mapillary.com>) have been using Mapzen Search to help with their crowdsourced mission of photographing every public place. Mapillary's amazing community of photomappers have biked, driven, and walked over 1 million kilometers taking photos of the world to build a crowdsourced "Street View". And searching those views is easier than ever, now that Mapzen Search powers the location search on their website and their **iPhone app** (<http://blog.mapillary.com/update/2015/09/16/new-iOS.html>)!



Gaia GPS

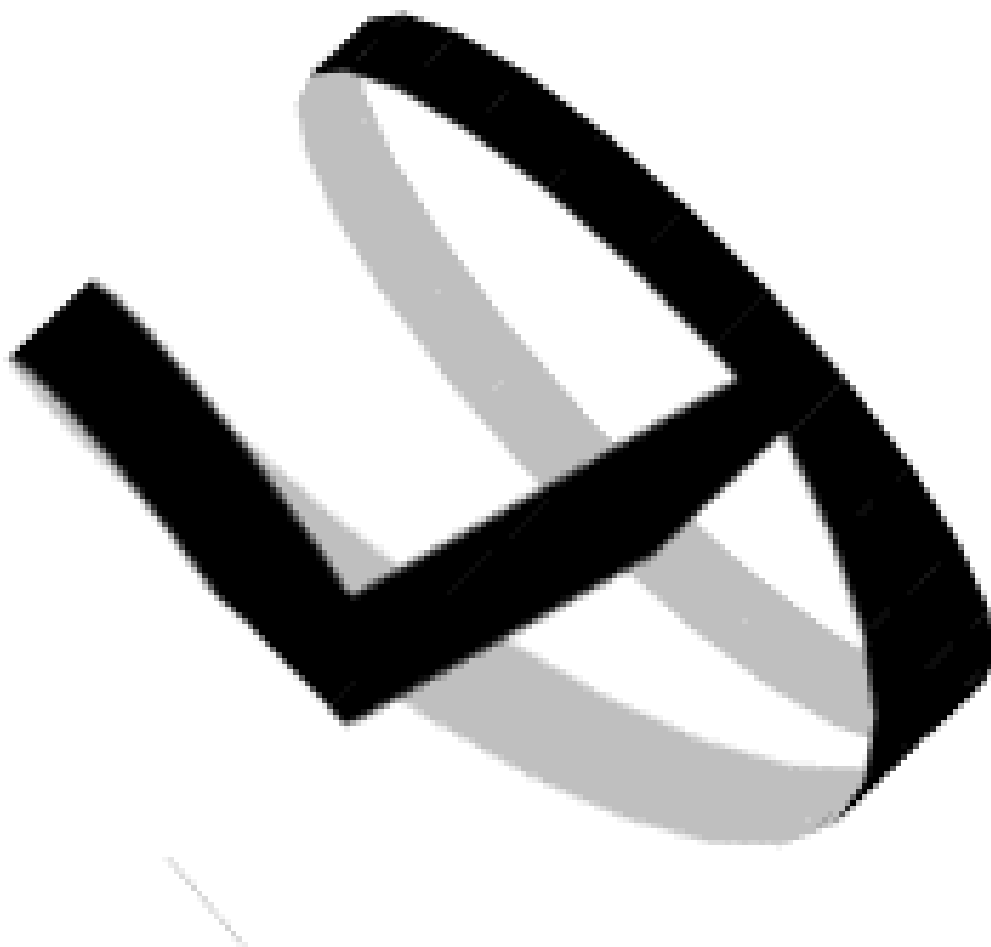
As one of the earliest adopters of Mapzen Search, **Gaia GPS** (<https://www.gaiagps.com>) has been pioneering new uses for the tool for months! Since we've been in beta, Gaia GPS has been helping hikers, bikers, and other outdoor explorers remember their outdoor adventures, using our **reverse geocoding** (<https://mapzen.com/documentation/search/reverse/>) to name their custom trails.

They just rolled the feature out into **their iOS app** (<https://www.gaiagps.com/apps/ios/>), so if you're planning an outdoor adventure and want to plan or log your journey, know that Mapzen Search is there to help you find yourself in GaiaGPS.

Meshu

Meshu (<https://meshu.io>) is artisanal map art jewelry commerce at its finest. Meshu is the brainchild of **Rachel Binx** (<http://rachelbinx.com/>), a self-described “data visualizer, art historian, mathematician, and GIF empress,” and lets people create custom jewelry from the geodata of places they’ve visited.

We seriously <3 Meshu (and its sister projects **monochōme** (<http://monochrome.com/>) and **TrekNotes** (<https://treknotes.co/>)) and love seeing our tools powering awesome map art. Make more!



CAD Mapper

CAD Mapper (<https://cadmapper.com>) is one of those remarkable small businesses that builds on top of OpenStreetMap and makes a point of giving back to the community. CAD Mapper allows architects and designers who work with CAD tools, like Autodesk’s venerable AutoCAD or

the open source QCAD, to pull custom geographic data from OpenStreetMap into their existing tools.

They've been using our autocomplete plugin for Leaflet to let users search for areas of interest to export.

CAD Mapper also **provides a CAD-friendly version of Metro Extracts**

(<https://cadmapper.com/#metro>), providing free AutoCAD DXF models of entire cities from OpenStreetMap!

Missed Your Project?

There have been hundreds of folks **who've given Mapzen Search**

(<https://mapzen.com/projects/search>) a try this year and we were only able to highlight a few of them. But if you've got a project we should know about, drop us a line and we'll try to showcase it in our next roundup.

Thanks so much for searching with us in 2015. We hope to find a lot more with you in 2016!

intro GIF via ***gifpop.io*** (<http://gifpop.io/pages/about-us>)

· 04 January 2016 ·



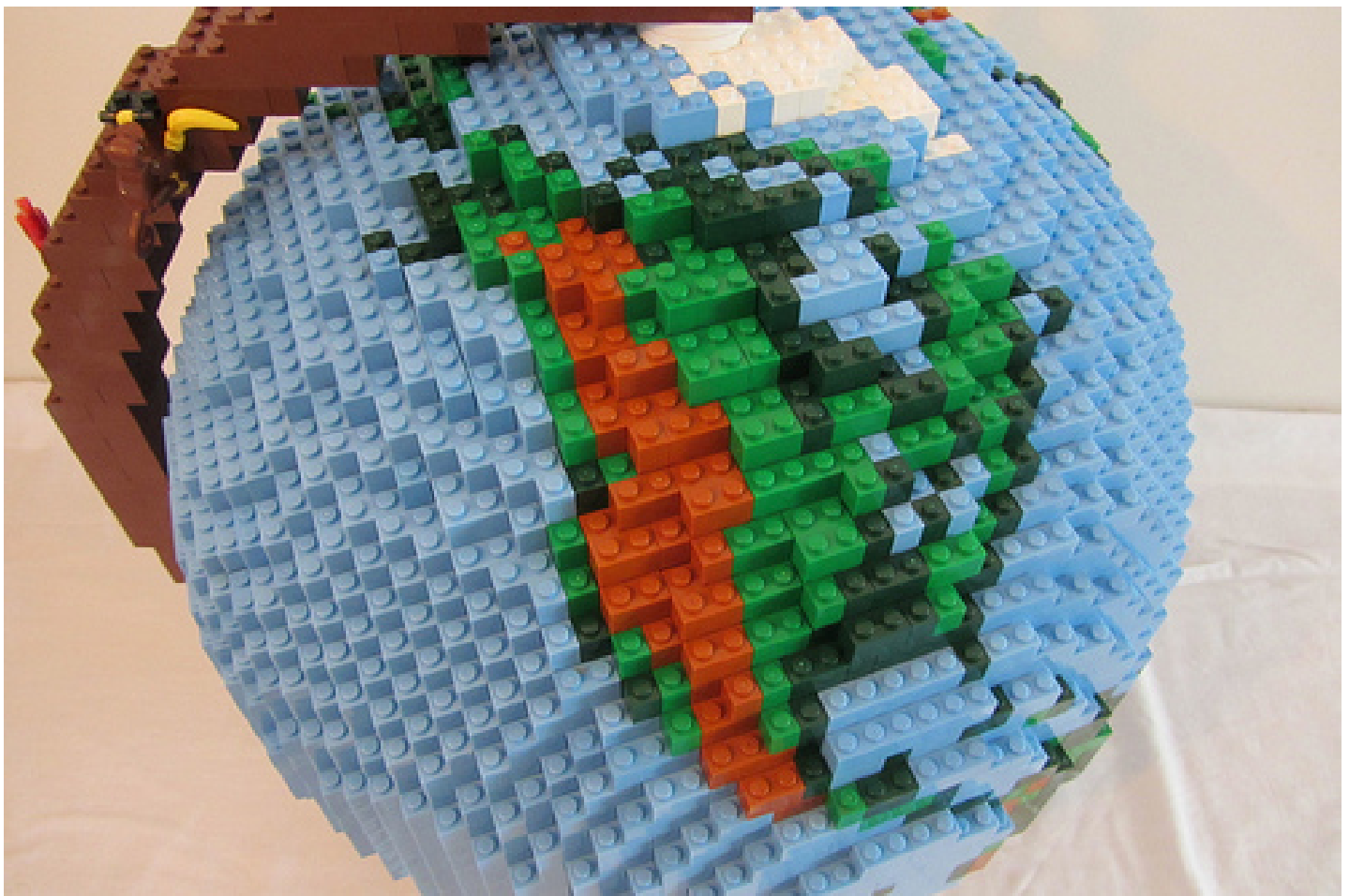
David Riordan

Former product manager for Mapzen Search. Historical mapping, open tools, open access, and open data.

© 2017 Mapzen

Mapzen at CES

This week we head to Las Vegas to welcome in the New Year at the world's largest technology trade show – the Consumer Electronics Show (CES)! We'll be with amongst over 170,000 attendees, 4000 exhibitors and 500 talks to revel in the possibilities of open mapping. From 3D visualizations to home monitoring, from satellite imagery to smart cities, from what's in your back pocket to your dream birthday gift, geospatial technologies continue to be a major theme for what it means to be in this digital world. And CES knows it.



If you are at the show, **tweet at us** (<https://twitter.com/intent/tweet?text=@Mapzen%20Hi!>) so we can meet up and talk about vector maps, search, routing and open data!

Lego globe images (<http://mocpages.com/moc.php/353076>) via **dirkb86** (<https://www.flickr.com/photos/dirkb86/albums/72157632691207934>), CC BY 2.0

· 05 January 2016 ·



Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.

© 2017 Mapzen

Mapzen "transpo" in DC

transitland (/tag/transitland) routing (/tag/routing)



Each January brings thousands of “transpo” professionals and enthusiasts to Washington, D.C. to discuss train schedules, road pavement, bridge engineering, and the many other intricacies of transportation. **Transportation Camp** (<http://transportationcamp.org/events/dc-2016/>) is an unconference that welcomes enthusiasts of all stripes, while the **Transportation Research Board’s annual meeting** (<http://www.trb.org/AnnualMeeting/AnnualMeeting.aspx>) is a more staid if also more massive meeting of professionals.

Mapzen will be at the TRB meeting to discuss the **Transitland open transit data service** (<https://transit.land>) and the **Mapzen Turn-by-Turn routing engine** (<https://mapzen.com/projects/valhalla>). We’ll be presenting at a TRB workshop called **Transformative Trends in Transit Data: General Transit Feed Specifications Bonanza** (<https://annualmeeting.mytrb.org/Workshop/Details/2446>) on Sunday, January 10. Bring a laptop and learn how to create transit data from our colleagues at **The World Bank** (<http://www.worldbank.org/en/topic/transport>), **Trillium Solutions**

(<http://trilliumtransit.com/>), **SUNY Albany** (<http://www.albany.edu/avail/>), **MIT** (<http://www.civicdatadesignlab.org/>), and **Azavea** (<http://www.azavea.com/>). (TRB registration is required to attend.)

And for a second year in a row, Mapzen is joining up with our colleagues at **Conveyal** (<http://conveyal.com/>) and **TransitScreen** (<http://transitscreen.com/>) to host a happy hour for mapping and “transpo” types of all sorts. All are welcome on the evening of Tuesday, January 12—TRB registration is not required to attend—but ***please RSVP if you’ll be joining us at the happy hour*** (<https://trbparty.splashthat.com/>).

Can’t make it to D.C. next week? Transportation Camp is popping up in at least **four more cities around the U.S. later in 2016** (<http://transportationcamp.org/>). And in the meantime, you’re invited to try out **Transitland** (<https://transit.land>) and **Mapzen Turn-by-Turn** (<https://mapzen.com/projects/valhalla>) here on the Internet.

photo by Jeremy Segrott (<https://www.flickr.com/photos/126337928@N05/18246320916/>), CC BY 2.0

· 05 January 2016 ·



Drew Dara-Abrams

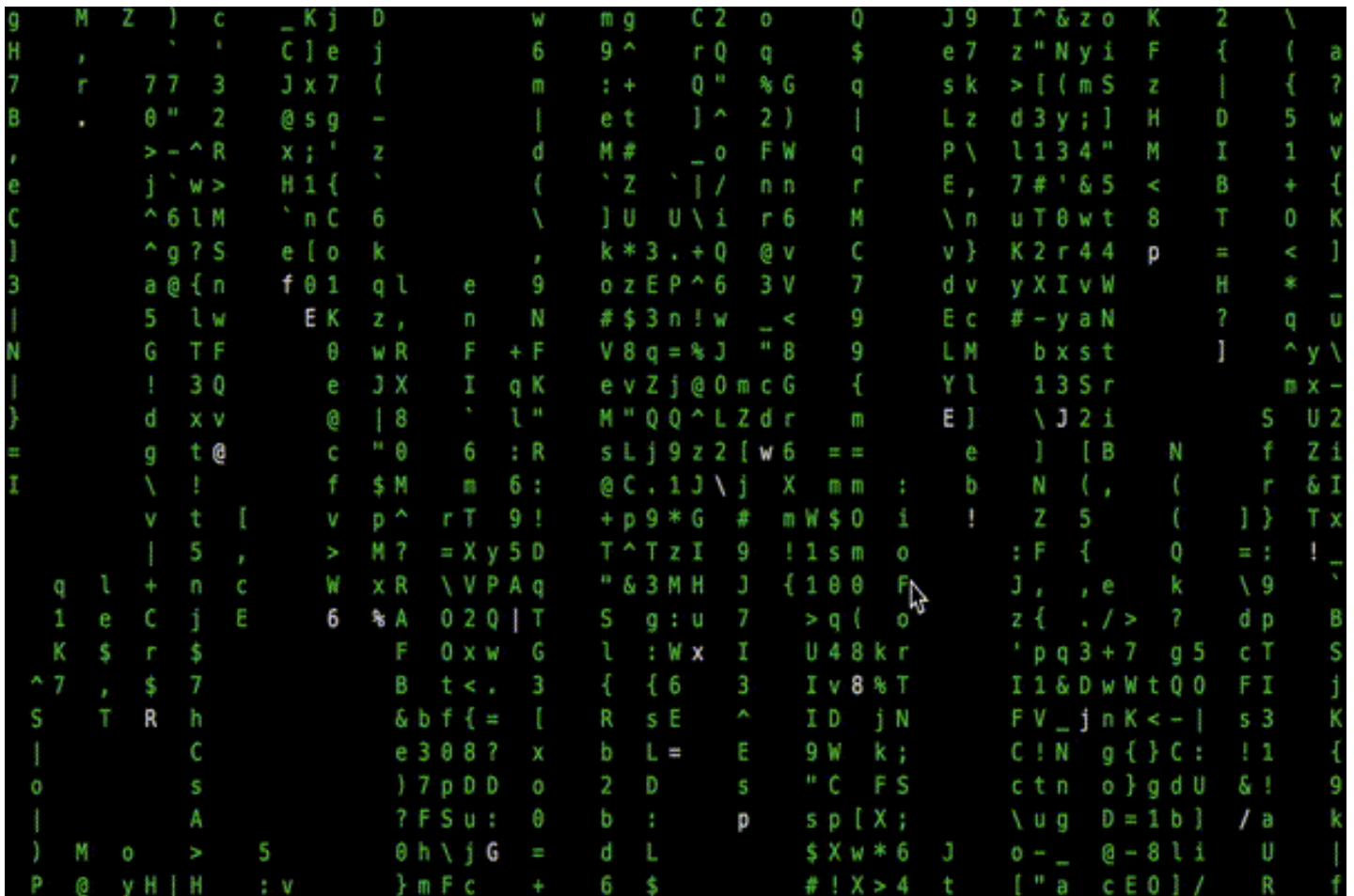
Drew leads Mapzen Mobility products (and aspires to being a flâneur).

© 2017 Mapzen

What We Talk About When We Talk About Engineering

engineering (/tag/engineering)

New Year's is a time to reflect on past successes and how to improve in the coming year. Such retrospection is probably not the first thing that comes to mind when thinking about engineers who, according to movies, ought to be staring at log lines whizzing down the screen at considerable pace.



made with cmatrix (<https://github.com/levithomason/cmatrix>)

Over the course of last few years, Mapzen engineers have amassed an amazing stack of systems and software that support our public facing projects and **APIs** (<https://mapzen.com/documentation/>). We will be **publishing a series of blog posts about**

our engineering practices (<https://mapzen.com/tag/engineering>), including what has succeeded and where we have failed. Here we'll cover various application frameworks. We'll go over AWS and the services we utilize in our attempt to reach what we like to refer to as 'AWS bingo'. We'll describe the way we manage infrastructure complexity and setup with Chef or Docker as well as some of the tooling we have built around it. We'll also discuss API management as well as metrics both for business analytics along with traditional elk stack deployment. We'll even talk about how we handle system outages and problems, although everything we do works perfectly all the time. ;)

Mapping is quite well covered on this blog, but this series is about the not actually mapping engineering (NAME). Our motivation is to discuss topics that are not necessarily related to Tangram, search, navigation or map tiles, but rather the cradle that supports them.

We deeply believe in open source and keep as much of our code open and available as we can. We've tried to maintain a standard where source code that addresses common problems is shared. Some things we do keep private, such as Mapzen-specific code or things that are undocumented, but in this series we will talk about them too.

We hope that by sharing our methods we can cut short someone's troubleshooting efforts when implementing large scale systems or building a mobile SDK. We would not be here without open source, and we want to be able to give back. You will be able to jump directly to this series via **/tag/engineering (<https://mapzen.com/tag/engineering>)**.

The first post by **Grant Heffernan (<https://github.com/heffergm>)** will be on **building a data pipeline to Elastic Search (</blog/mapzen-search-data-pipeline>)** to keep Mapzen Search data fresh and available.

· 07 January 2016 ·



Baldur Gudbjornsson

Former VP of Engineering, with special focus on devops, infrastructure, community and api management.

© 2017 Mapzen

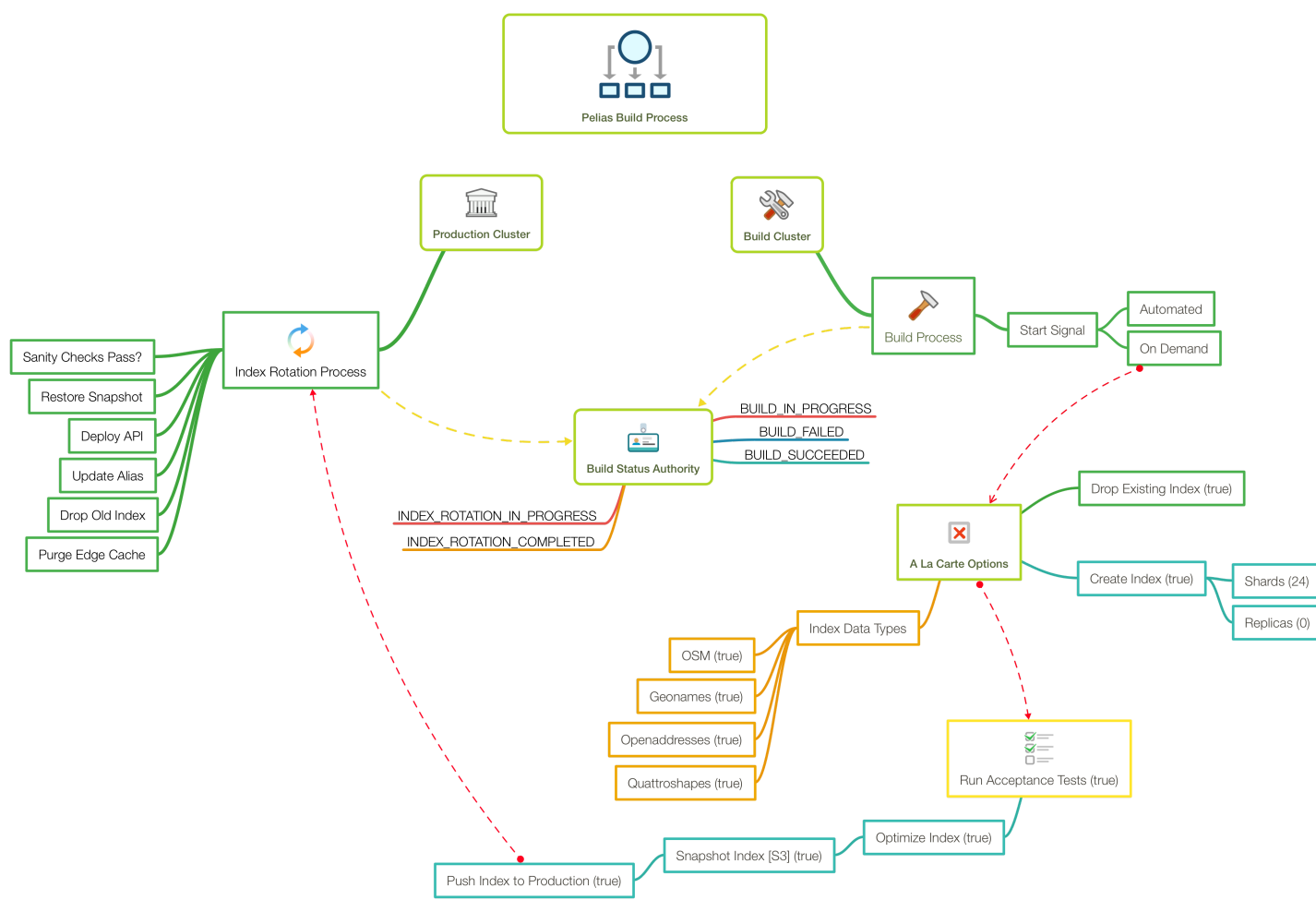
Mapzen Search Data Pipeline

engineering (/tag/engineering) search (/tag/search)

This is the first in a **series of posts about engineering at Mapzen** (<https://mapzen.com/tag/engineering>) – learn more here (/blog/engineering-series).

Operationally, Mapzen Search is comprised at a very basic level of an API and an ElasticSearch cluster. Where things get complicated is in the building of a pipeline that keeps the data in that cluster both up to date and highly available.

Overview



*The build pipeline in all its glory (actually, somewhat simplified for **easier viewing** (https://mapzen-assets.s3.amazonaws.com/images/mapzen-search-data-pipeline/pelias_build_process_transparent.png))*

Arriving at the solution you see outlined above has been an iterative process. When we started working on Search over two years ago, simply indexing all of OSM took around 30 days. That made building any sort of automated build pipeline difficult, and in any event wasn't on the immediate to-do list. Now, we index geonames, quattros shapes, osm and openaddresses in a little under two days. As the product has matured, so has the automation.

How Does It Work?

Underlying Services

Search is hosted in AWS, and we leverage a service called **Opsworks** (<https://aws.amazon.com/opsworks/>). Every week, an automated process initiates a call to the API underlying that service, which kicks off a build. That build is run on a dedicated environment comprised of an Elastic cluster, the Pelias API and a **Dashboard** (<http://pelias-dashboard.mapzen.com/pelias>).

Import Performance

The dashboard is simply a nice way for us to visually watch the build process and to share that status with users. The process itself, which is crafted in chef, makes use of **GNU Parallel** (<http://www.gnu.org/software/parallel/>) to run several imports in tandem. This speeds things along substantially, as opposed to running each import serially. How many imports you can run in parallel will depend to a large extent on the hardware you're using, both for the Elastic cluster and for the system on which the import process itself is running. We recommend using SSD volumes for both the data you're importing (for example, an OSM pbf file), as well as for the leveldb volume for the **Address Deduplicator** (<https://github.com/pelias/address-deduplicator>) if you're doing (you guessed it) address deduplication as part of your import.

Testing & Backup

When the import completes, provided all the jobs ran successfully, we move on to **acceptance testing** (<https://github.com/pelias/acceptance-tests>). This is a crucial step in validating the data we just indexed as fit to move to production or not. Provided these tests pass and no regressions are detected, the index is optimized (critical for good performance!) and then a

snapshot is taken. The snapshot is stored on S3 to allow us to easily retrieve old snapshots for testing, backup, etc., as well as to allow easy access from multiple environments where we might want to make use of it (e.g. production).

Roll to Prod

In the event there was a failure upstream, we notify interested parties and we can investigate the failure. Otherwise, at this point, the build itself is complete! Now we can move on to getting that index into production. Again, this is accomplished by leveraging the underlying Opsworks API. A request is made that results in a chef recipe being run against the production Elastic cluster. The recipe, in this case, executes some rather lengthy code I won't bore you with. But in essence it does the following:

- checks for the availability of a new snapshot
- if available, performs a number of sanity checks on our cluster and the snapshot to ensure we can safely proceed
- loads the snapshot into the cluster
- updates the index alias to reference the new index
- drops the old index
- issues a purge request to the edge cache

Summary

And that, friends, is how we keep our Pelias data fresh and do so with a minimum of manual intervention. If you have questions about any of the specifics, get in touch!

· 07 January 2016 ·



Grant Heffernan

Grant is a sysadmin with fingers in all of Mapzen's pies. Egli non si senta italiano, ma per fortuna o purtroppo...

Targeted Editing – Name that Building

targeted-editing (</tag/targeted-editing>) **osm** (</tag/osm>)

Maps are current as of Oct 2016.

Named buildings are easier to find!

Welcome back to our **Targeted Editing** series where today you can put on a sweater, or make **OpenStreetMap** (<http://www.openstreetmap.org/>) data better! (Both take about the same amount of time.)

Read on for some basic uses for building names, and where you can check to see if you can use your **local knowlege** to add some names to buildings where they are needed.

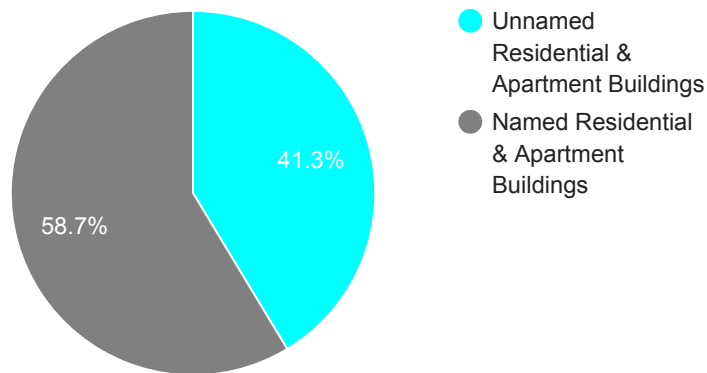
How can building names be used?

1. **Help with search.** Addresses are great, but a building name can take you a long way towards finding where you want to go. Apartment complexes usually have names. Office buildings sometimes have names.
2. **Help with routing.** Routing and search often work together. If you would like to find a route to a building, but you don't know the address, the name might help.
3. **Help with labeling.** The more you zoom in, the more labels you might see, but not if the features don't have names.

We are missing some building names here!

There are many different types of buildings, but check out this pie chart for residential & apartment buildings in Hong Kong:

Hong Kong, RESIDENTIAL & APARTMENT Buildings Without Names



Residential (<https://wiki.openstreetmap.org/wiki/Tag:building%3Dresidential>) & **apartment** (<https://wiki.openstreetmap.org/wiki/Tag:building%3Dapartments>) building names are very common in Hong Kong. Are they common where you live, too?

How you can help improve buildings without names:

Here is a map styled by **Peter Richardson** (<https://github.com/meetar>) & **Nathaniel V. Kelso** (<https://github.com/nvkelso>) showing many residential & apartment buildings – check out the **styling filters** (<https://github.com/mapzen-data/targeted-editing/blob/gh-pages/te-building-names/map/buildings.yaml>) we use to highlight residential buildings with and without names. To add a name to a nameless residential building **that should have one**, hover over those that are bright blue to bring up an info bubble with links to editing tools.



[Leaflet](#) | Geocoding by [Mapzen](#)

(This map is interactive! Open full screen ↗ (<http://mapzen-data.github.io/targeted-editing/te-building-names/map/?buildings>))

Not familiar with Hong Kong? Search or pan over to your home town to contribute your local knowledge to the map. You will see your changes right away in OpenStreetMap and in **Mapzen Vector Tiles** (<https://mapzen.com/projects/vector-tiles>) within an hour, including the map right here on this page!

Note that if residential buildings are only tagged with `building=yes`, they may not be highlighted on this map even if they have names. If you do want to tag these as `building=residential` or `building=apartment`, shift-click on the map to edit in iD. You can learn much more about building tag use at **taginfo** (<https://taginfo.openstreetmap.org/keys/building#values>).

Need instructions on how to edit with iD? Here are some links to outstanding tutorials from LearnOSM, the OpenStreetMap wiki, and the United States Department of State's Humanitarian Information Unit:

- **LearnOSM** (<http://learnosm.org/en/>)
- **OSM Beginners' Guide** (http://wiki.openstreetmap.org/wiki/Beginners'_guide)
- **MapGive Learn to Map** (<http://mapgive.state.gov/learn-to-map/>)

Are you a mapping wiz and interested in a more advanced editor? Try out **JOSM** (<https://josm.openstreetmap.de>) with **excellent documentation** (<https://www.mapbox.com/blog/osm-mapping-guide/>) from Mapbox.

Thanks, and please check back soon for the next post in our series!

All the posts in the Targeted Editing series:

- **Airports** (<https://mapzen.com/blog/targeted-editing-airport-polygons>)
- **Banks** (<https://mapzen.com/blog/new-years-resolutions-money/>)
- **Building Names** (<https://mapzen.com/blog/targeted-editing-name-that-building>)
- **Campus Mapping** (<https://mapzen.com/blog/targeted-editing-campus-mapping/>)
- **Cycleways** (<https://mapzen.com/blog/targeted-editing-cycleways/>)
- **Fitness** (<https://mapzen.com/blog/new-years-resolutions-fitness>)
- **Groceries** (<https://mapzen.com/blog/new-years-resolutions-groceries>)
- **Hospitals** (<https://mapzen.com/blog/targeted-editing-hospital-polygons>)
- **Schools** (<https://mapzen.com/blog/targeted-editing-school-polygons>)
- **Stadiums & Parking** (<https://mapzen.com/blog/targeted-editing-tailgate-mania>)
- **Street Names** (<https://mapzen.com/blog/targeted-editing-no-name-roads>)
- **Transit Colours** (<https://mapzen.com/blog/targeted-editing-transit-colours>)
- **Travel & Lodging** (<https://mapzen.com/blog/new-years-resolutions-travel>)

· 08 January 2016 ·



Indy Hurt

Indy was our resident data scientist lending her geographic expertise to all things "open".

© 2017 Mapzen

Lines

tangram (/tag/tangram) cartography (/tag/cartography)

Cartographic design is all about drawing. Before I begin something new, I always start by reflecting on the essentials. I like to dissect and think about the most basic elements. What's in my toolbox? What are my ingredients? Let's think about lines. **What can lines describe?**

A line can describe a route – from point A to point B



A line can describe a shape or a form, such as a coastline



(<https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/airports.yaml#8/37.686/-122.478>)

open the **map** (<https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/airports.yaml#8/37.686/-122.478>), and take a look at the **code** (<https://github.com/tangrams/multiverse/blob/gh-pages/styles/airports.yaml#L13-L43>) that makes it work

A line can describe a ground feature such as these airport runways at SFO



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

In this case it's a drawing but, now also a symbol, like a signature or calligraphic writing. Fascinating!

map (<https://tangrams.github.io/tangram-frame?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/airports.yaml#13/37.6189/-122.3764>) | code (<https://github.com/tangrams/multiverse/blob/gh-pages/styles/airports.yaml#L59-L93>)

With contour lines we can give forms dimension

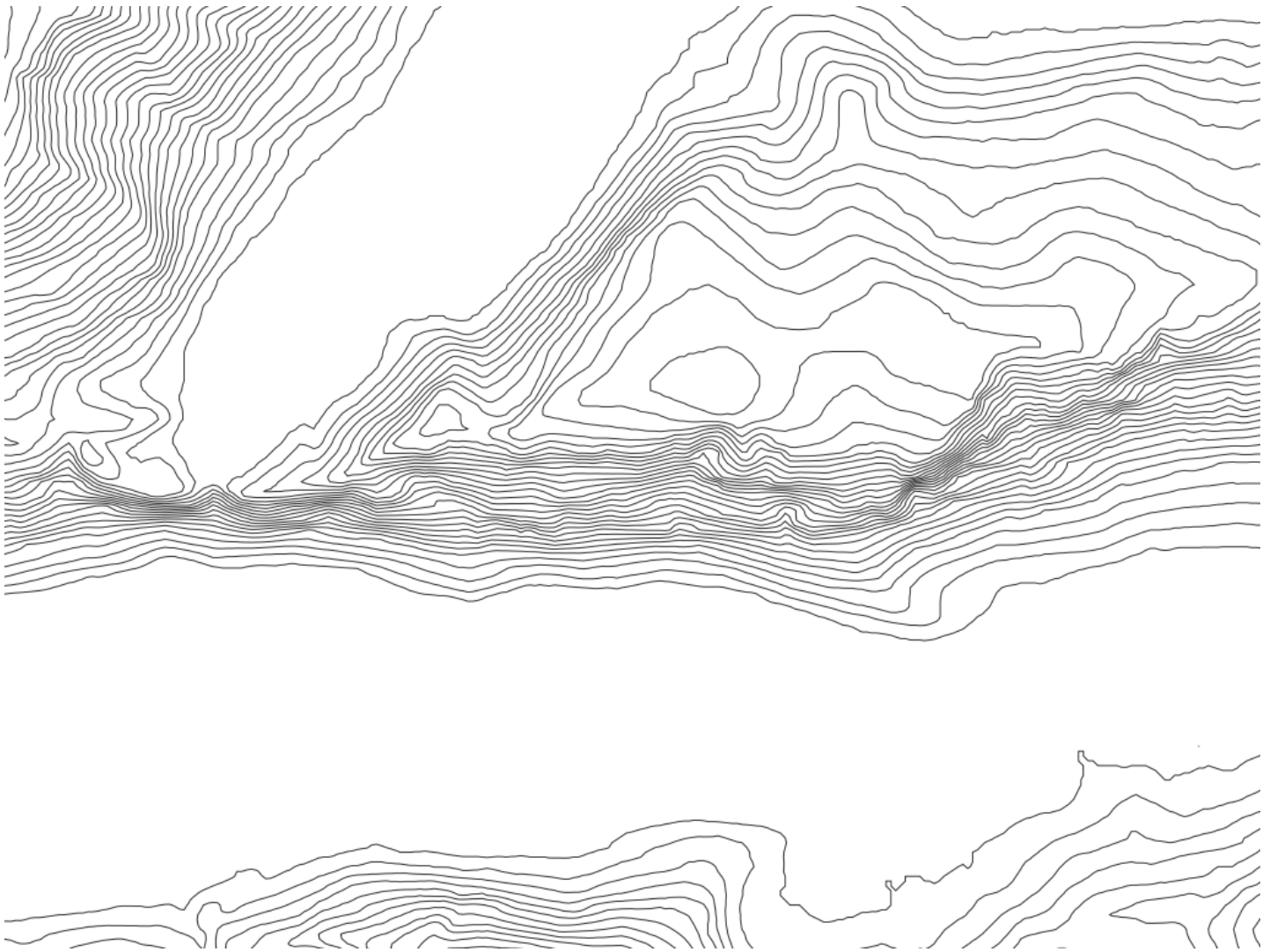
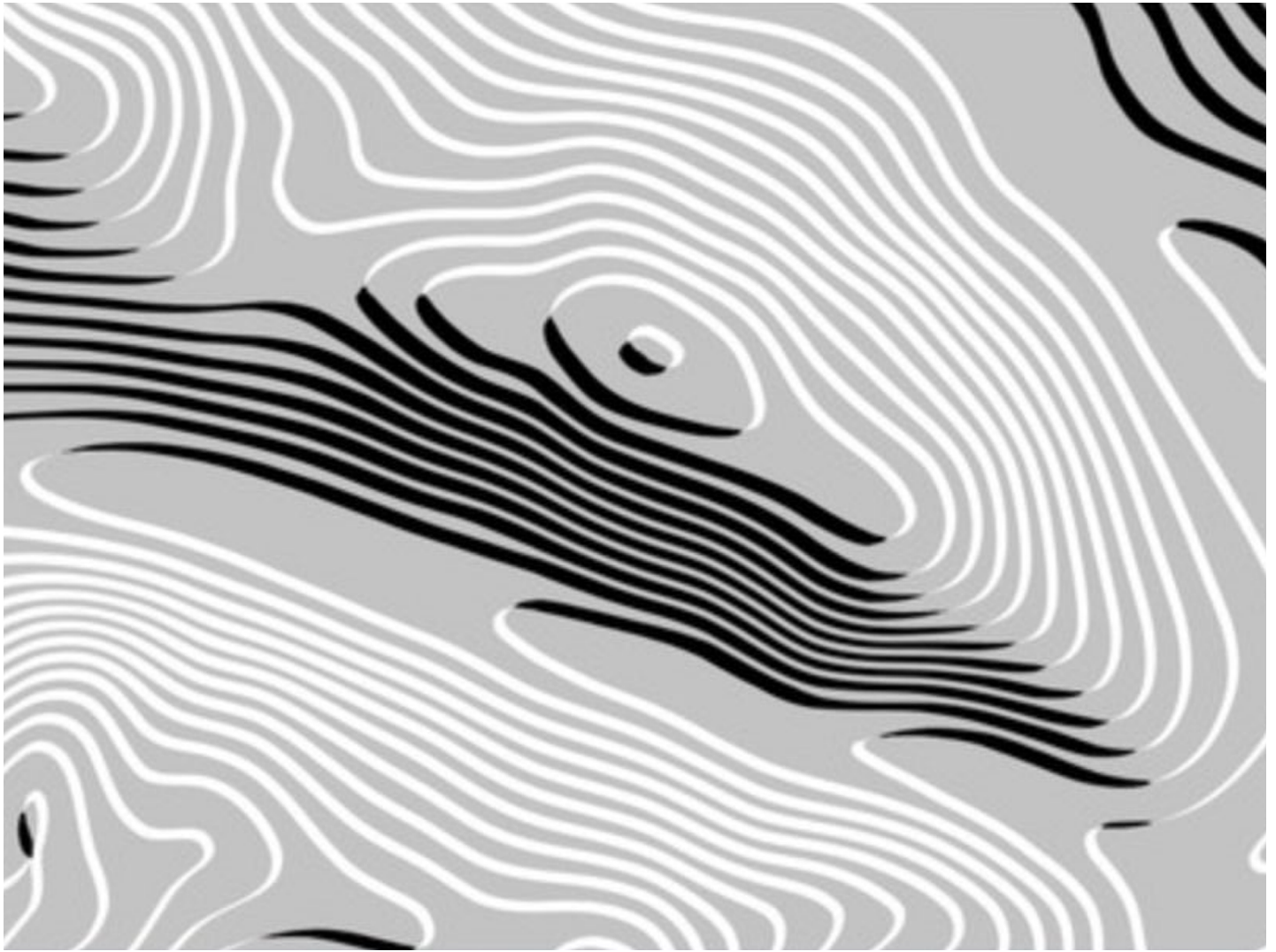


image from *Thematic Mapping* - **link** (<http://blog.thematicmapping.org/2012/07/creating-contour-lines-with-gdal-and.html>)

Here's a drawing technique I learned at NACIS, called Illuminated or Shadowed Contours, or otherwise called "Tanaka Contours"

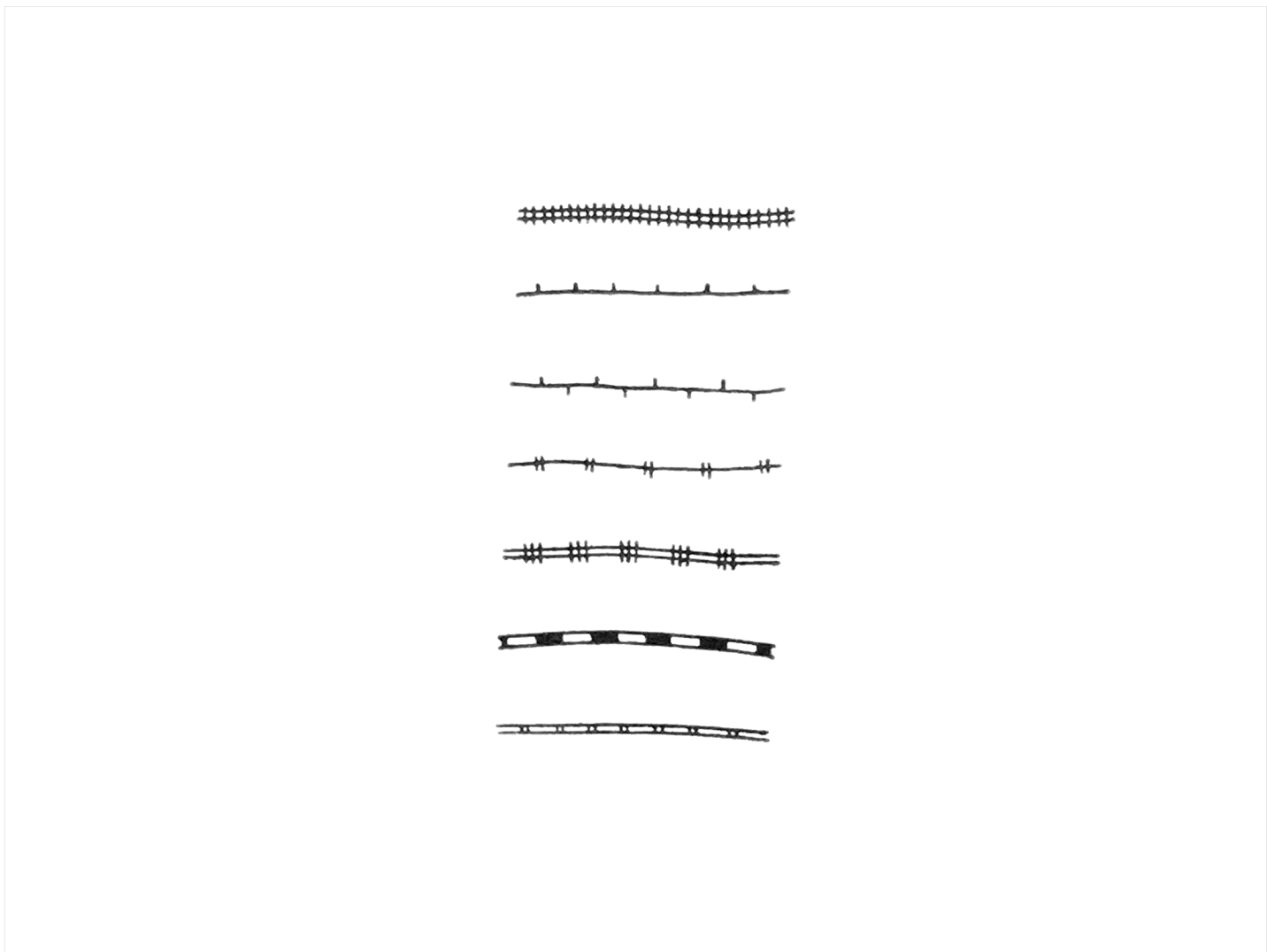


(<https://speakerdeck.com/nvkelso/evaluating-the-effectiveness-of-illuminated-and-shadowed-contour-lines>)

James Eynard gave a detailed talk about this drawing technique in Minneapolis at NACIS this year, Evaluating the Effectiveness of Illuminated and Shadowed Contour Lines - [link](#)

(<https://speakerdeck.com/nvkelso/evaluating-the-effectiveness-of-illuminated-and-shadowed-contour-lines>). This technique was developed by Tanaka Kitiro in 1950 - [link](http://wiki.gis.com/wiki/index.php/Tanaka_contours) (http://wiki.gis.com/wiki/index.php/Tanaka_contours).

Here are a variety of railroad lines by cartographer Erwin Raisz (<http://www.raiszmaps.com>)



Lines can also describe a structure such as a grid



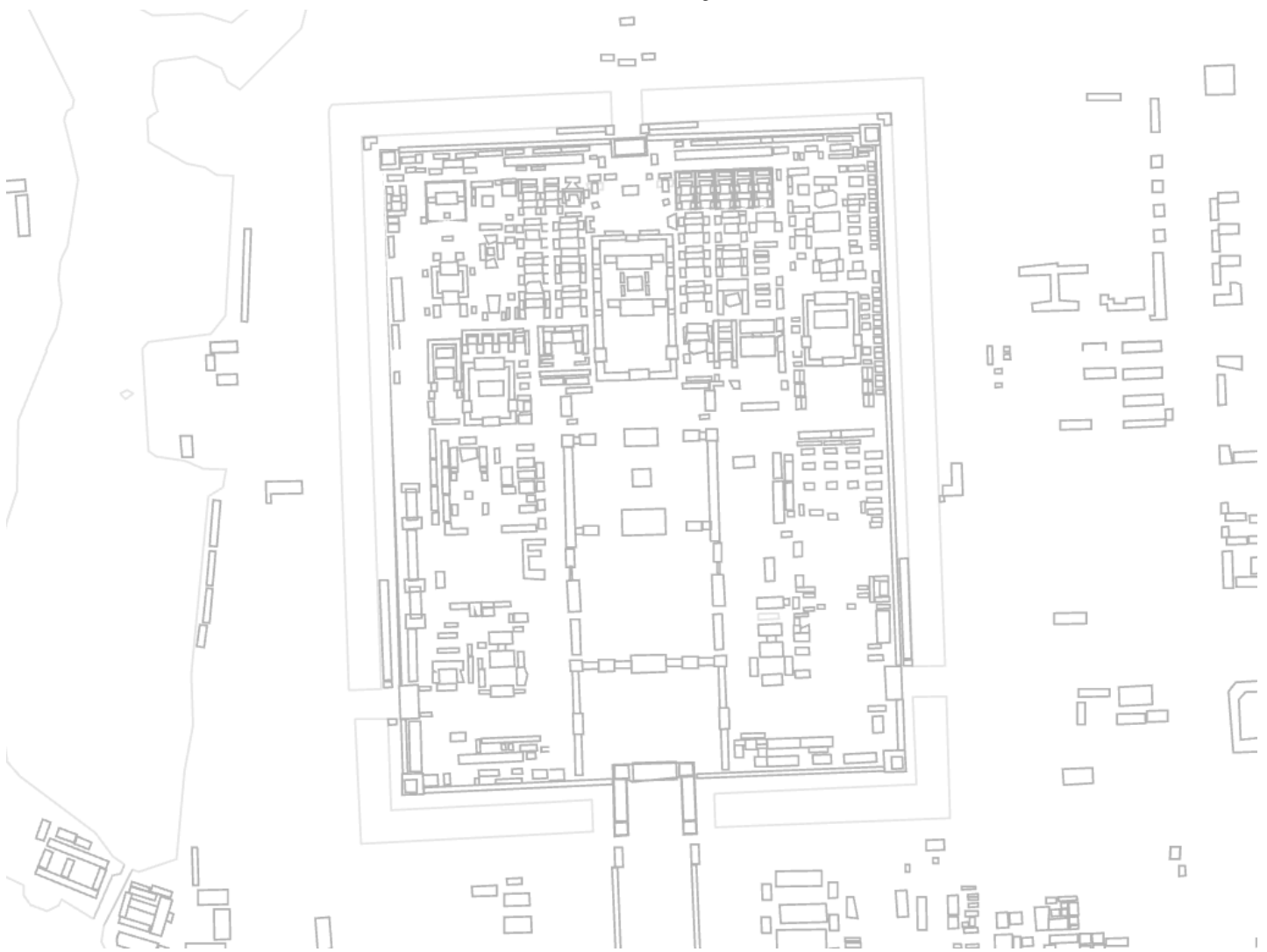
This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

Check out these **grids** (<http://usshop.gestalten.com/grid-index.html>)!

map (<https://tangrams.github.io/tangram-frame?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/tile-grid.yaml#5/21.398/-157.939>) | **code** (<https://github.com/tangrams/multiverse/blob/gh-pages/styles/tile-grid.yaml#L9-L52>)

Lines can form into polygons such as these building footprints



([https://tangrams.github.io/tangram-frame/?](https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-footprint.yaml#16/39.9159/116.3904)

[url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-footprint.yaml#16/39.9159/116.3904](https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-footprint.yaml#16/39.9159/116.3904))

Forbidden City, Beijing

map ([https://tangrams.github.io/tangram-frame/?](https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-footprint.yaml#16/39.9159/116.3904)

[url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-footprint.yaml#16/39.9159/116.3904](https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-footprint.yaml#16/39.9159/116.3904)) | code

(<https://github.com/tangrams/multiverse/blob/gh-pages/styles/building-footprint.yaml>)

Apply a grid to a building facade and it becomes a surface



(<https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-grid.yaml#18/40.76789/-73.98202>)

Columbus Circle, Manhattan

map (<https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-grid.yaml#18/40.76789/-73.98202>) | code
(<https://github.com/tangrams/multiverse/blob/gh-pages/styles/building-grid.yaml#L13-L29>)

Repetition! The tool I love most! Repeat a line and we have stripes!



(<https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-stripes.yaml#19/-15.78948/-47.89438>)

Brasília

map (<https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-stripes.yaml#19/-15.78948/-47.89438>) | code
(<https://github.com/tangrams/multiverse/blob/gh-pages/styles/building-stripes.yaml#L9-L23>)

Change the rhythm and we have a gradient



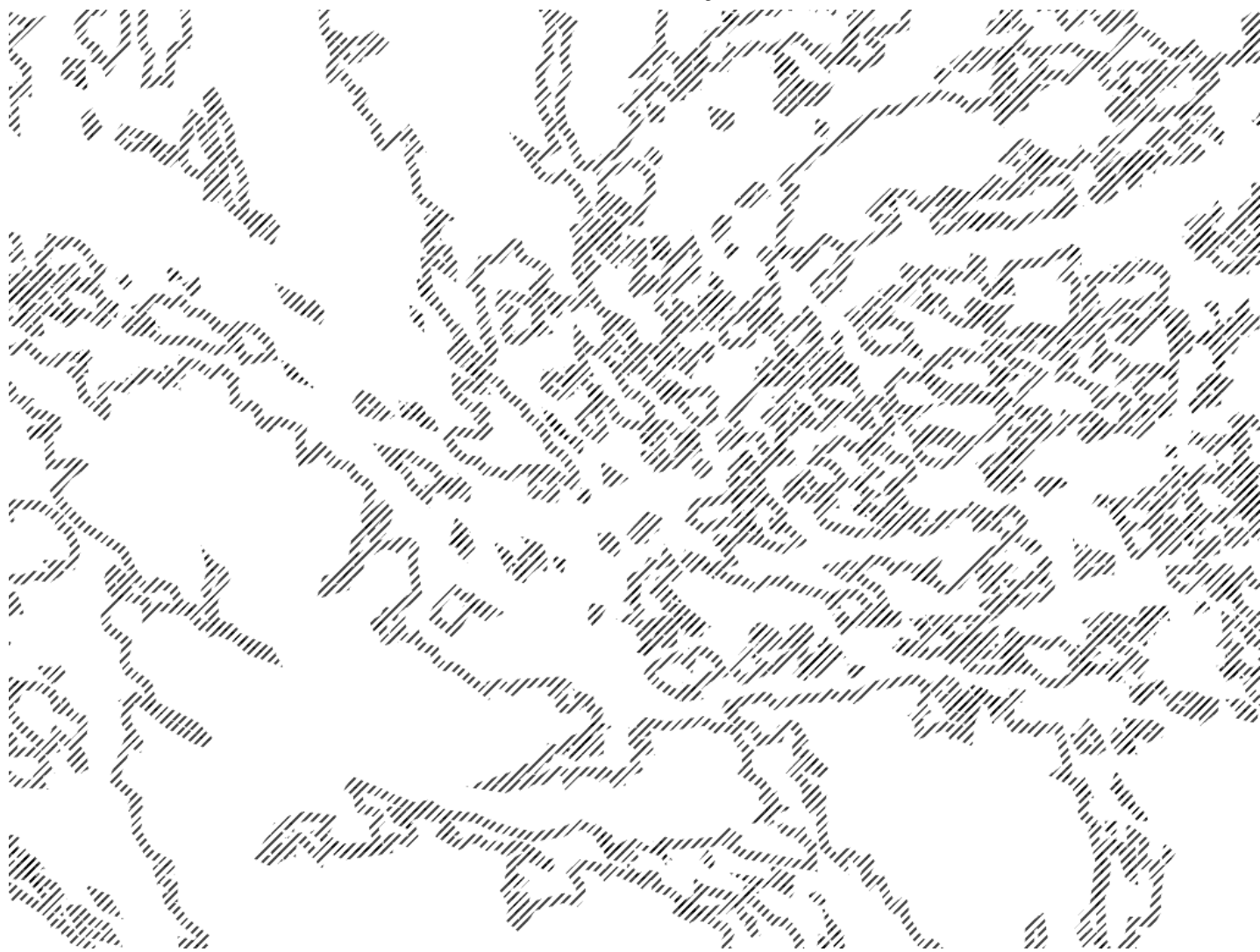
This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

Lower Manhattan

map (<https://tangrams.github.io/tangram-frame?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-linegradient.yaml#18/40.70825/-74.00822>) | code (<https://github.com/tangrams/multiverse/blob/gh-pages/styles/building-linegradient.yaml#L7-L34>)

Stripes can become texture when applied to lines, such as these diagonal stripes on a coastline



([https://tangrams.github.io/tangram-frame/?](https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/coastline-stripes.yaml#11/59.4065/18.5576)

[url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/coastline-stripes.yaml#11/59.4065/18.5576](https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/coastline-stripes.yaml#11/59.4065/18.5576))

Stockholm Coastline Area

map ([https://tangrams.github.io/tangram-frame/?](https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/coastline-stripes.yaml#11/59.4065/18.5576)

[url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/coastline-stripes.yaml#11/59.4065/18.5576](https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/coastline-stripes.yaml#11/59.4065/18.5576)) | code

(<https://github.com/tangrams/multiverse/blob/gh-pages/styles/coastline-stripes.yaml#L8-L39>)

Or apply these repeated wave lines to an area and you can create a water texture



([https://tangrams.github.io/tangram-frame/?](https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/water-waves.yaml#5/7.667/-243.369)

[url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/water-waves.yaml#5/7.667/-243.369](https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/water-waves.yaml#5/7.667/-243.369))

Southeast Asia

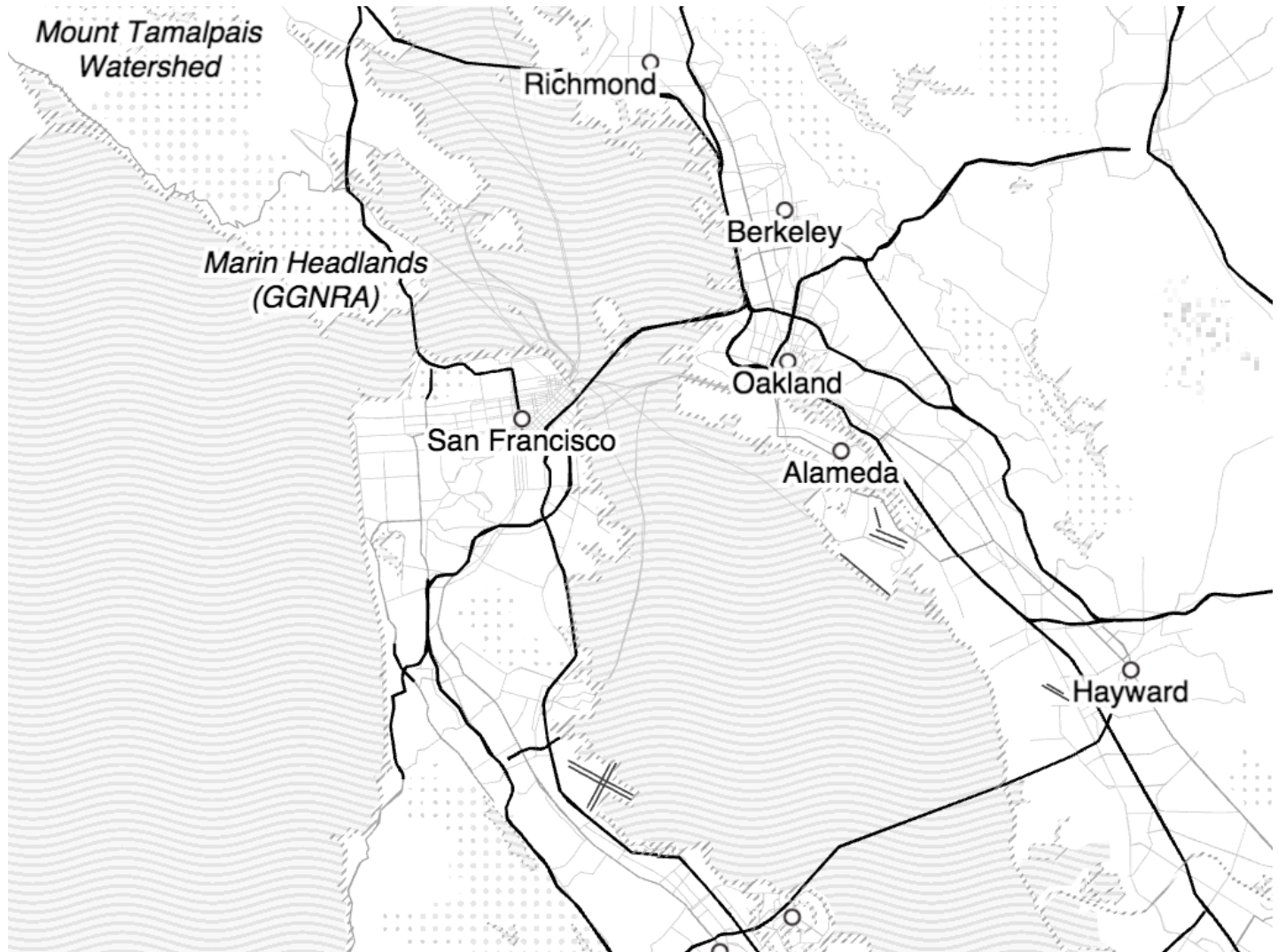
map ([https://tangrams.github.io/tangram-frame/?](https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/water-waves.yaml#5/7.667/-243.369)

[url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/water-waves.yaml#5/7.667/-243.369](https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/water-waves.yaml#5/7.667/-243.369)) | code (<https://github.com/tangrams/multiverse/blob/gh-pages/styles/water-waves.yaml#L8-L36>)

I use some of these drawing techniques in one of our featured styles called **Refill** (<https://tangrams.github.io/refill-style/#15/37.8044/-122.2708>). Refill is a continuation of the **Toner** (http://content.stamen.com/dotspotting_toner_cartography_available_for_download) style

I started at **Stamen** (<http://stamen.com>). Think of Refill as a more detailed Toner, an elaboration with the GL capabilities of Tangram. With Tangram I'm able to explore more detailed line work, patterns and building extrusions. Below are some highlights of line-work from the Refill Style along with some isolated examples of drawing techniques I talked about above.

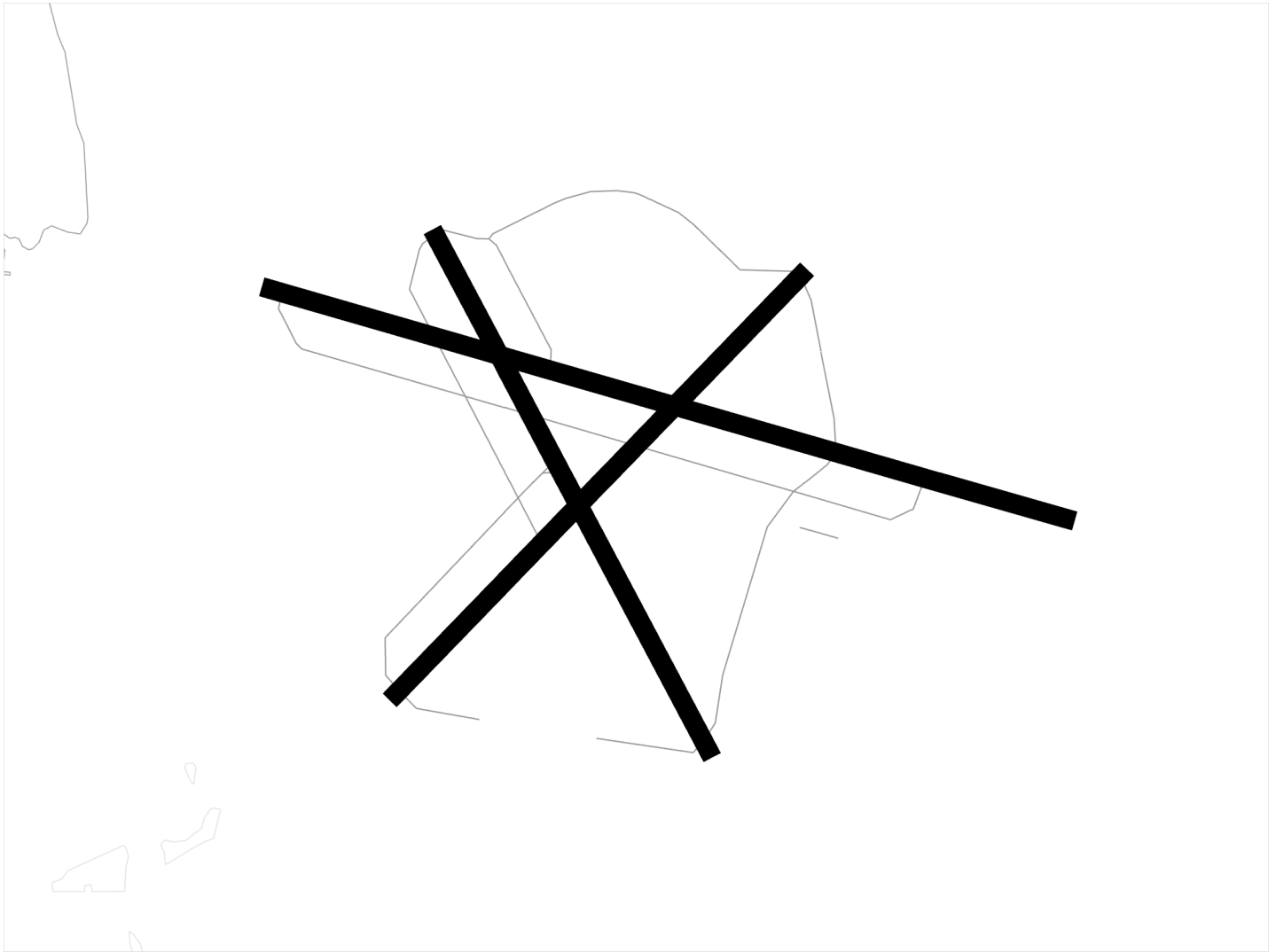
San Francisco in Refill



(<https://tangrams.github.io/refill-style/#11/37.7588/-122.3187>)

map (<https://tangrams.github.io/refill-style/#11/37.7588/-122.3187>) | code
(<https://github.com/tangrams/refill-style>)

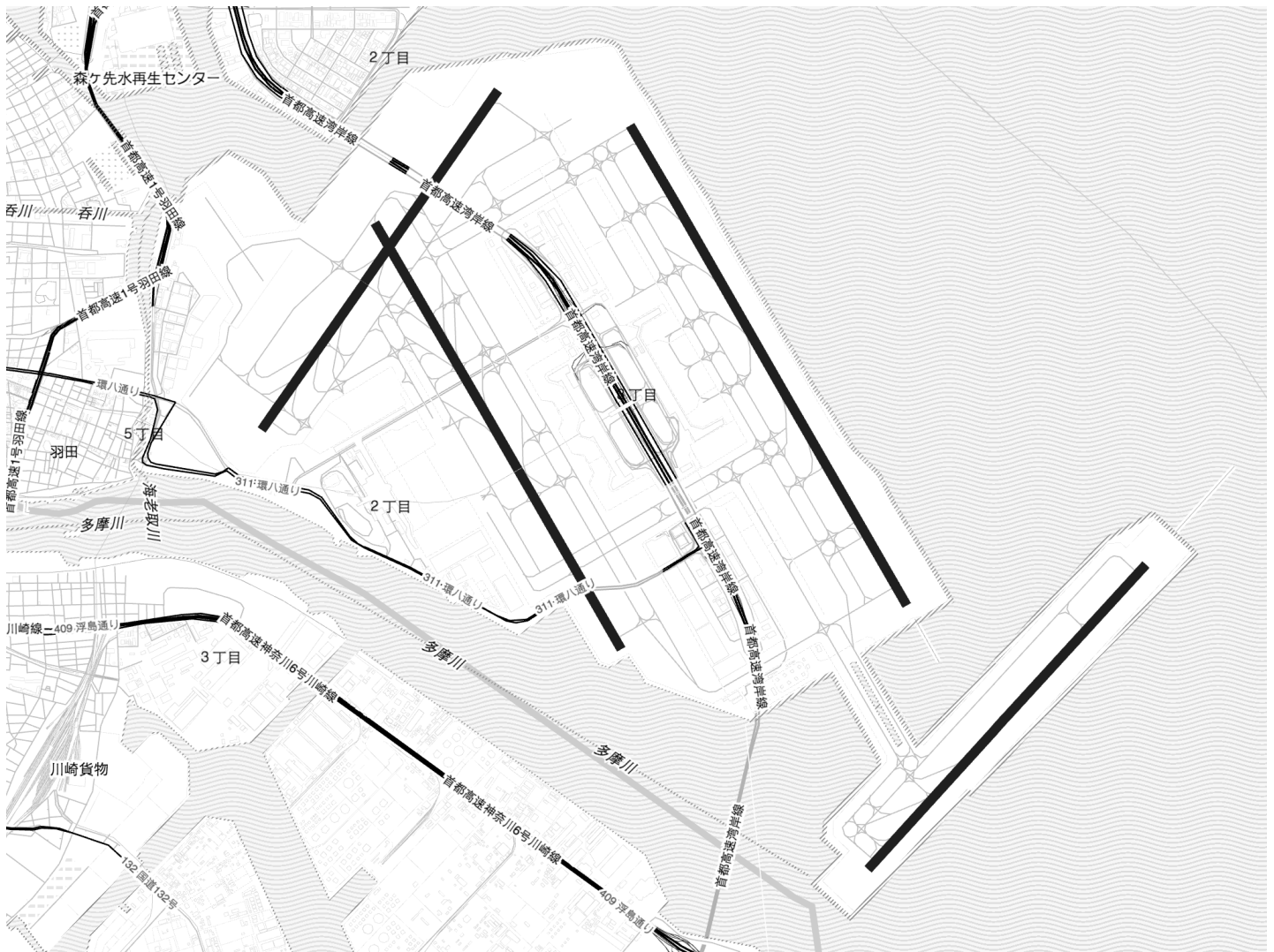
YYJ Victoria, British Columbia



(<https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/airports.yaml#14/48.6467/-123.4274>)

map (<https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/airports.yaml#14/48.6467/-123.4274>) | code
(<https://github.com/tangrams/multiverse/blob/gh-pages/styles/airports.yaml#L79-L88>)

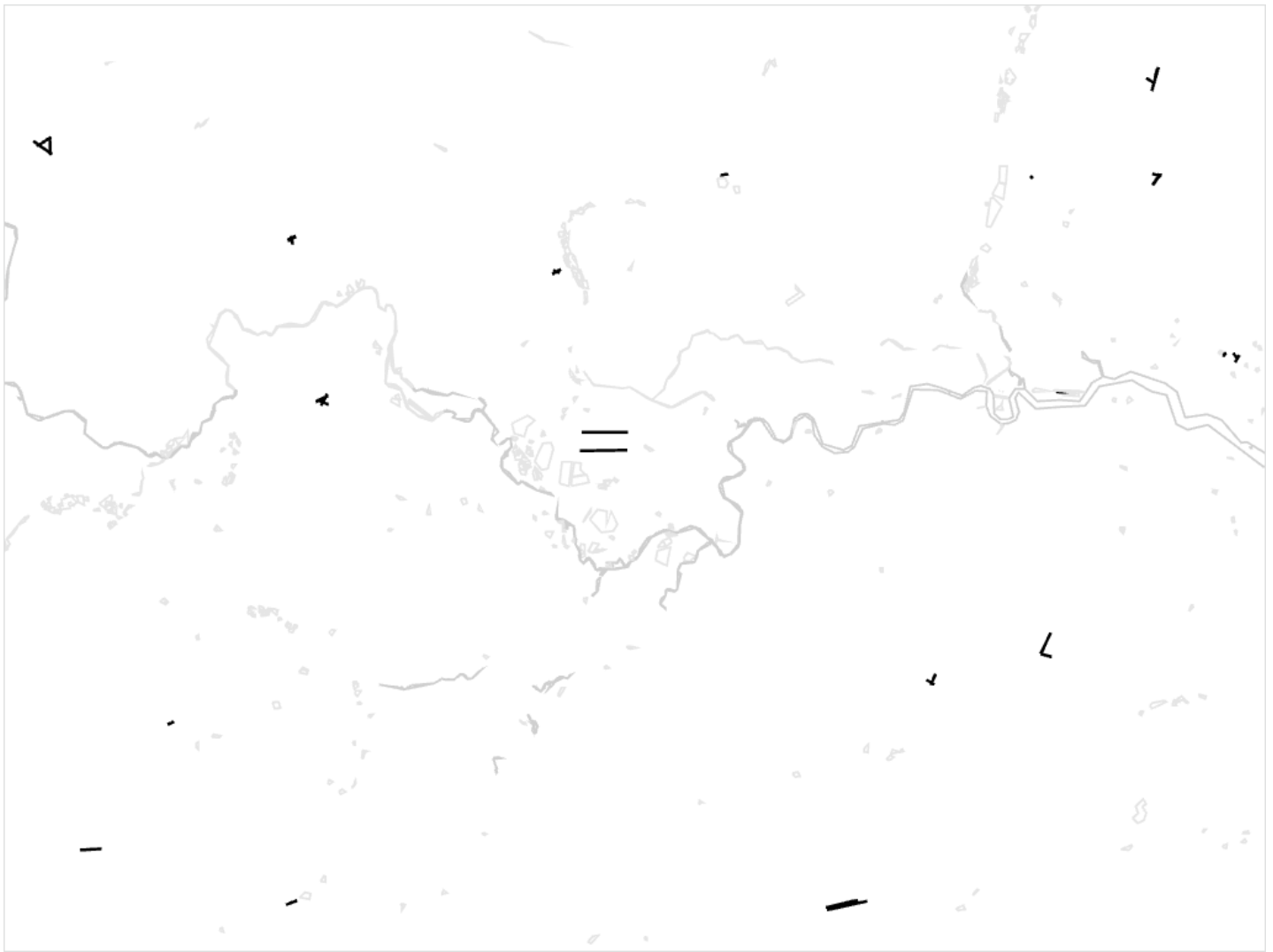
HND Tokyo in Refill



(<https://tangrams.github.io/refill-style/#14/35.5468/139.7858>)

map (<https://tangrams.github.io/refill-style/#14/35.5468/139.7858>) | code
(<https://github.com/tangrams/refill-style>)

Airport Runways Looking Like Symbols around London



(<https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/airports.yaml#10/51.5433/-0.0941>)

map (<https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/airports.yaml#10/51.5433/-0.0941>) | code
(<https://github.com/tangrams/multiverse/blob/gh-pages/styles/airports.yaml#L66>)

Shinjuku West



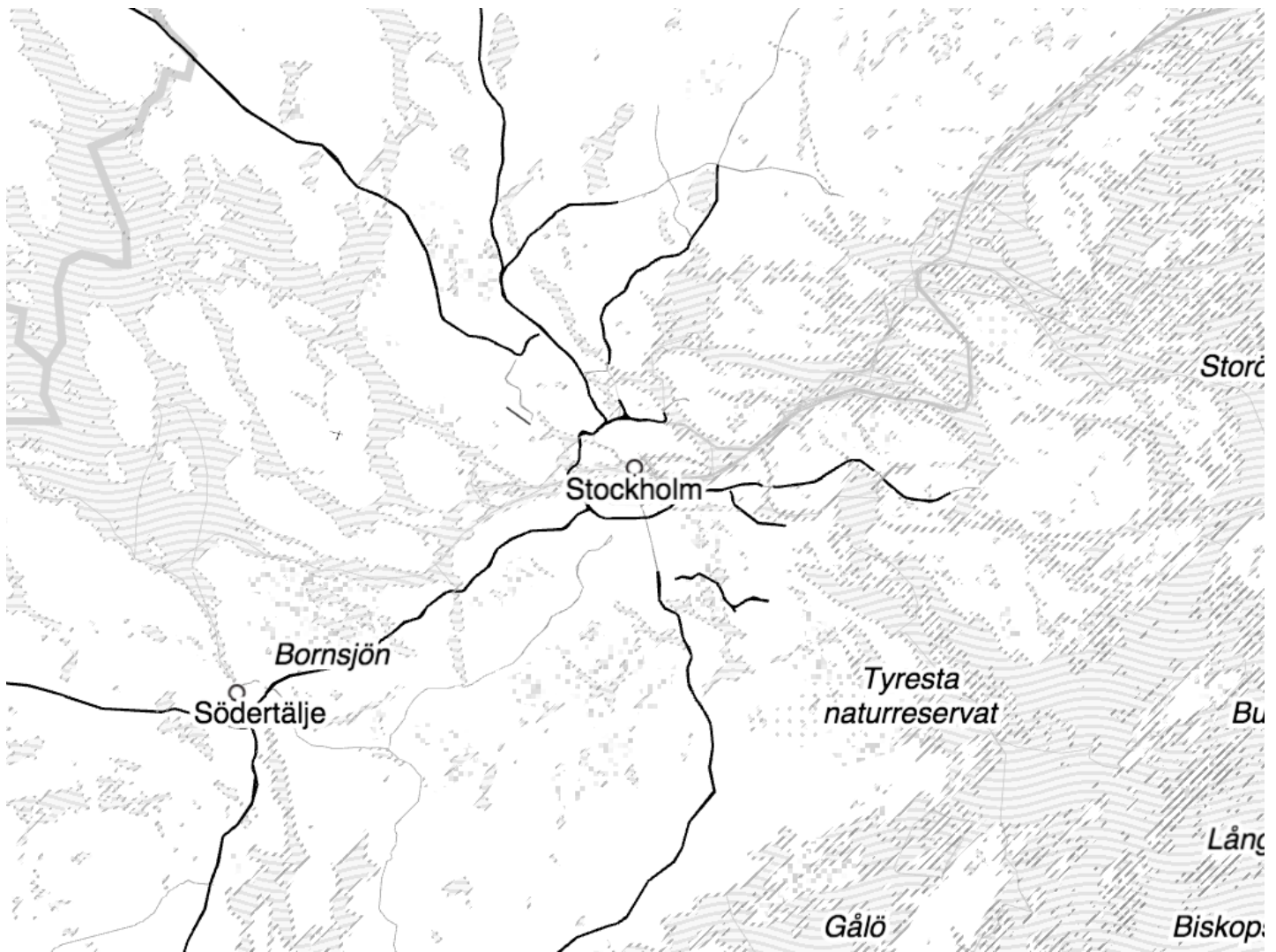
([https://tangrams.github.io/tangram-frame/?](https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-grid.yaml#16/35.6900/139.6920)

[url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-grid.yaml#16/35.6900/139.6920](https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-grid.yaml#16/35.6900/139.6920))

map ([https://tangrams.github.io/tangram-frame/?](https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-grid.yaml#16/35.6900/139.6920)

[url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-grid.yaml#16/35.6900/139.6920](https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-grid.yaml#16/35.6900/139.6920)) | code (<https://github.com/tangrams/multiverse/blob/gh-pages/styles/building-grid.yaml>)

Stockholm Coastline Views in Refill



(<https://tangrams.github.io/refill-style/#10/59.3062/18.0711>)

map (<https://tangrams.github.io/refill-style/#10/59.3062/18.0711>) | code
(<https://github.com/tangrams/refill-style>)

Rome in Stripes!

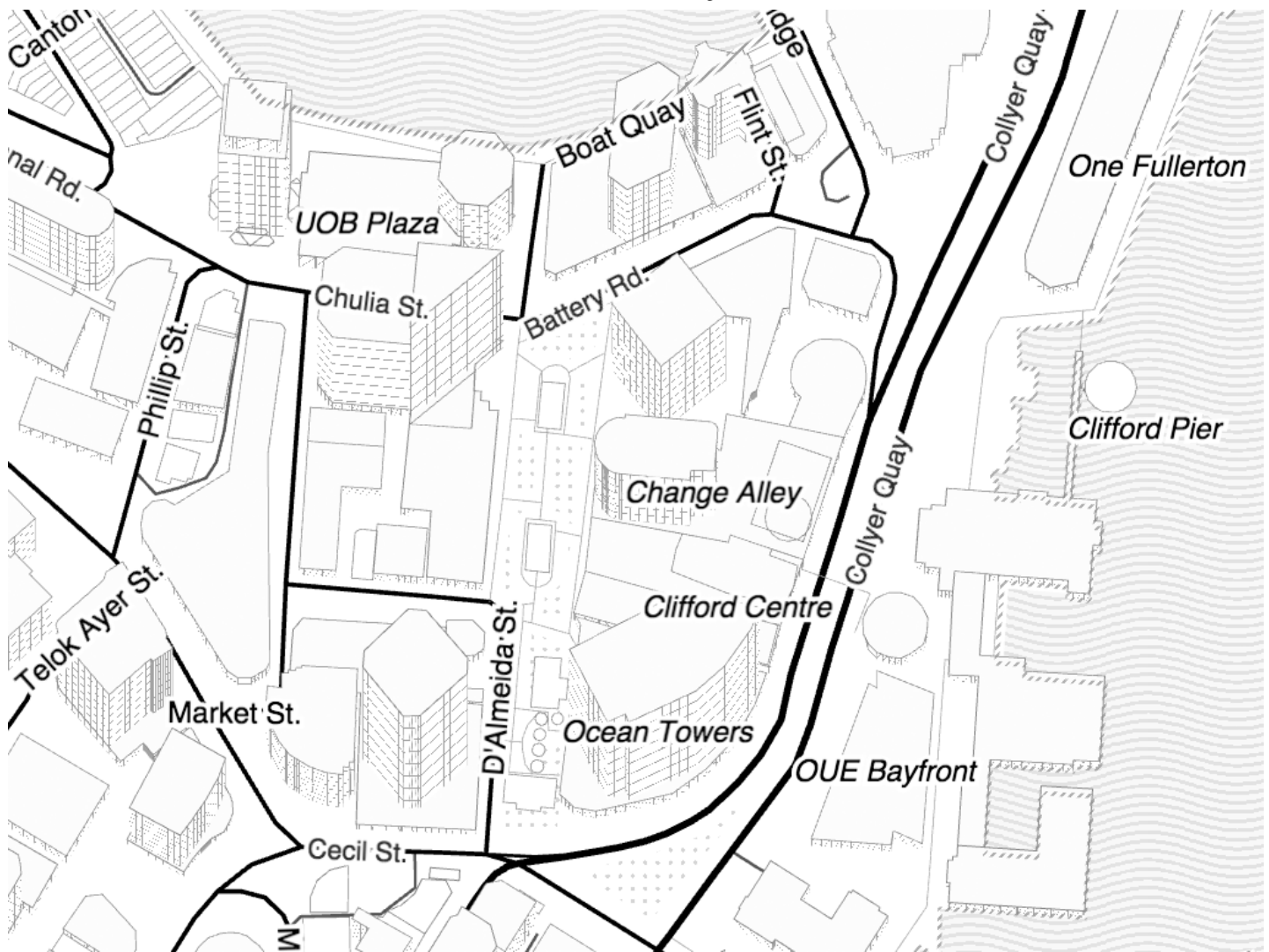


This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

map (<https://tangrams.github.io/tangram-frame/?url=https://raw.githubusercontent.com/tangrams/multiverse/gh-pages/styles/building-stripes.yaml#19/41.89244/12.48525>) | code (<https://github.com/tangrams/multiverse/blob/gh-pages/styles/building-stripes.yaml>)

Building extrusions in Singapore



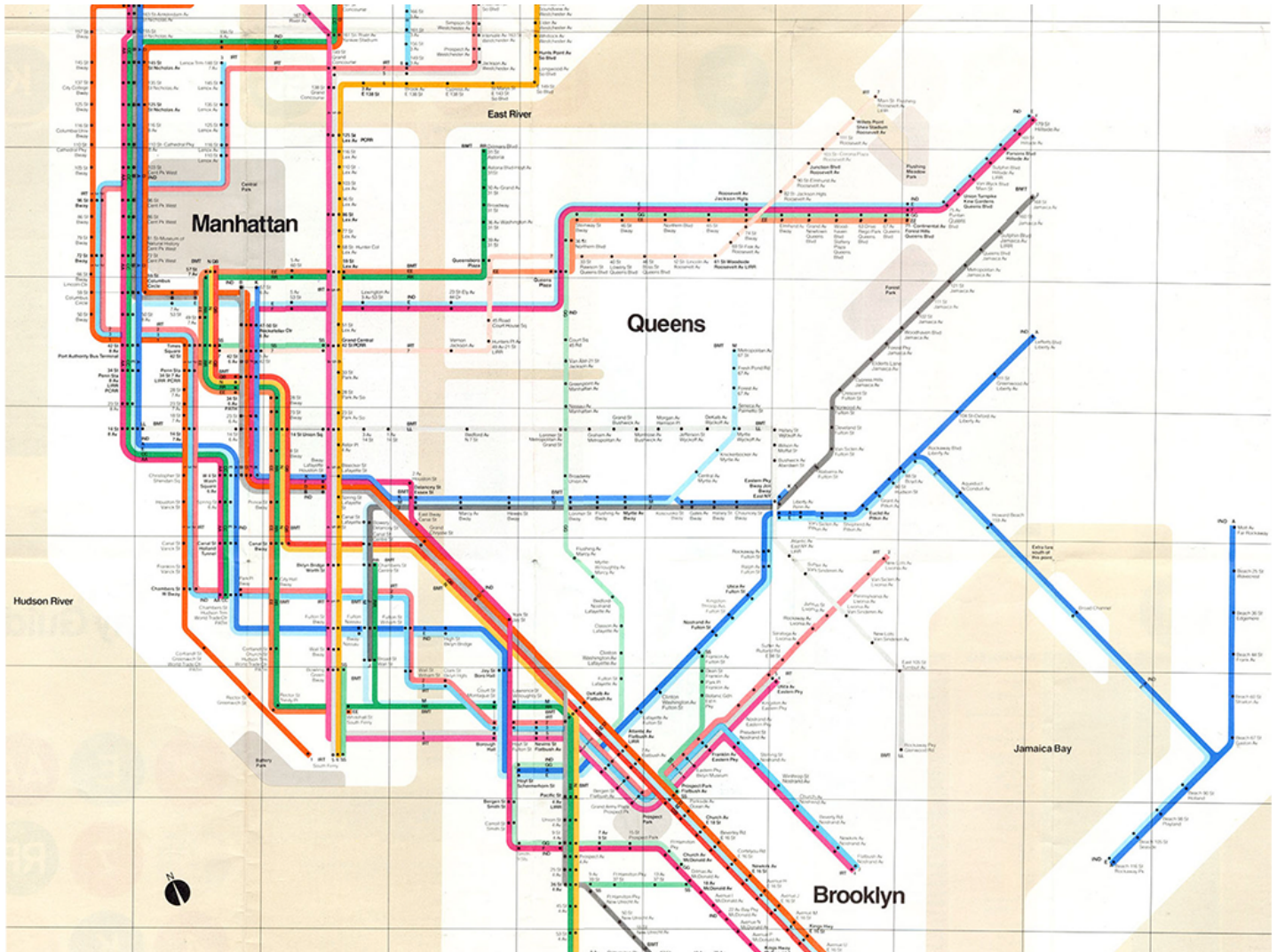
(<https://tangrams.github.io/refill-style/#18/1.28414/103.85171>)

map (<https://tangrams.github.io/refill-style/#18/1.28414/103.85171>) | code
(<https://github.com/tangrams/refill-style>)

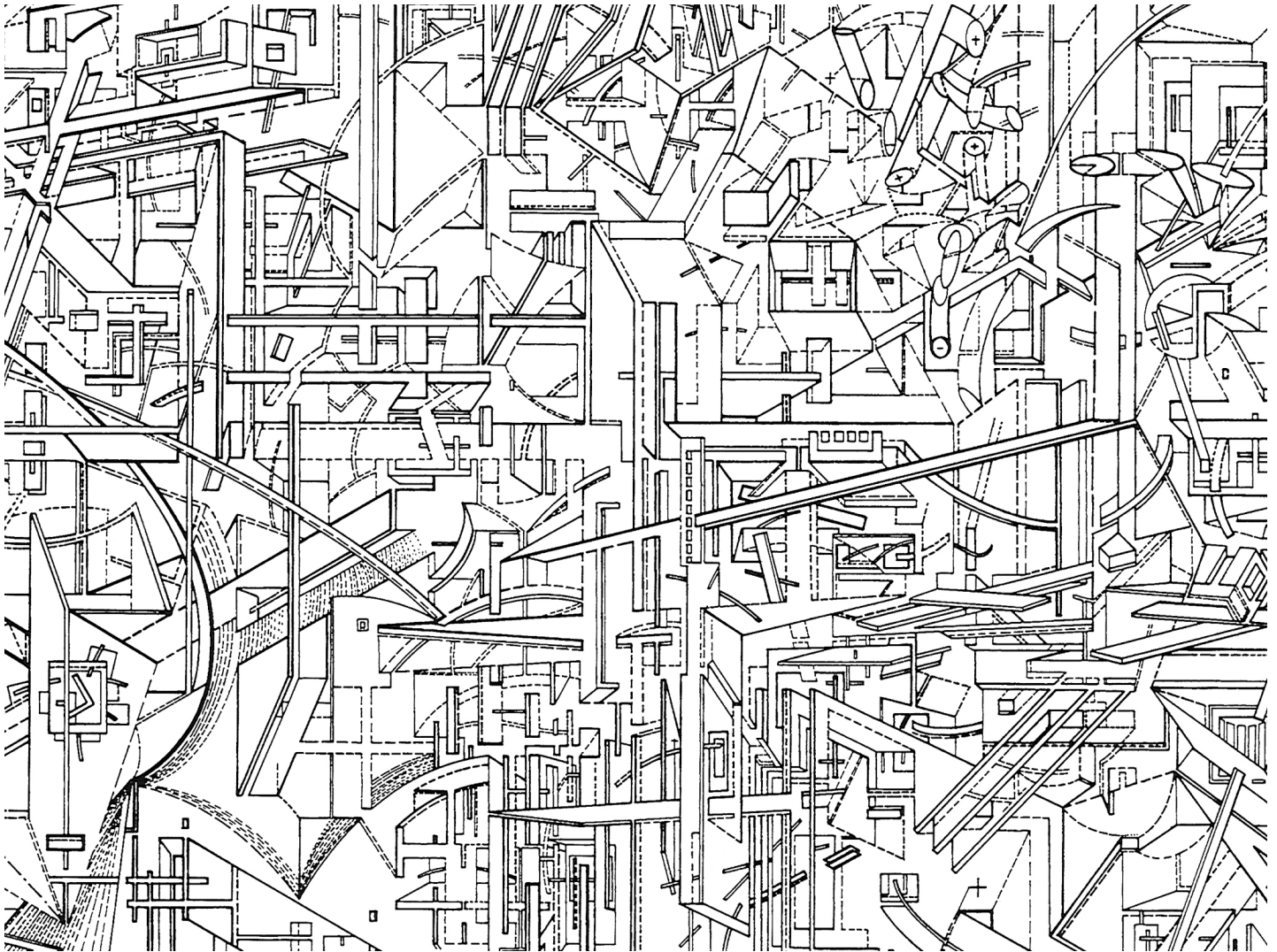
I'll conclude this with some gorgeous examples of beautiful line-work for your inspiration.

[illegible]

Massimo Vignelli (https://en.wikipedia.org/wiki/Massimo_Vignelli)



Daniel Libeskind (<http://libeskind.com/work/micromegas/>)



Julie Mehretu (<http://www.art21.org/artists/julie-mehretu>)



Go and explore what you can do with lines!

Read the **Tangram Documentation** (<https://mapzen.com/documentation/tangram/Scene-file/>) for more details. **Let us know** (<https://twitter.com/intent/tweet?text=@Mapzen%20I%20love%20lines!>) if you have any questions!

· 14 January 2016 ·



Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

© 2017 Mapzen

New Year's Resolutions – Groceries

targeted-editing (/tag/targeted-editing) **osm** (/tag/osm)

Maps are current as of Oct 2016.

Were you supposed to be eating healthier in 2016? Have you fallen into the snack wagon? Eating healthy is always high up on the list of the most popular resolutions every year. Goals need details to stick, and OpenStreetMap needs details to grow. Let's combine the two by adding and enhancing grocery stores in OpenStreetMap. Eating healthy often involves eating more fruit and vegetables. Where do you find those? In the grocery store!

Business listings and other points of interest are slowly gaining momentum in OpenStreetMap. Curious to know how prevalent grocery stores and similar businesses are in OpenStreetmap? Check out this table which includes a dozen cities known for tempting cuisine:

	Grocery Points of Interest*	with Addresses	with Website	with Hours
Auckland	342	6%	1%	1%
Dublin	758	14%	4%	6%
Hong Kong	783	5%	2%	6%
Melbourne	1,624	13%	43%	5%
Mexico City	881	7%	1%	5%
Mumbai	150	7%	1%	1%
San Francisco	763	47%	17%	15%
São Paulo	1,046	22%	6%	3%
Seoul	2,292	2%	0.2%	1%
Singapore	415	11%	6%	5%

	Grocery Points of Interest	with Addresses	with Website	with Hours
Stockholm	1,005	16%	14%	28%
Vancouver	203	39%	10%	12%

*SQL queries (https://github.com/mapzen-data/targeted-editing/blob/gh-pages/queries/grocery_sql.sql) for those interested in generating stats for additional cities.

Seoul leads the pack with the most features where one might find groceries, but lags behind other metros like Melbourne, San Francisco, and Stockholm which include more detailed tags on each store. Lots of opportunities to enhance existing features and add missing ones!

Helpful Tags

If your personal food pyramid is not what you'd like it to be, get inspired by adding or enhancing grocery related businesses in OpenStreetMap! Need some help choosing tags? Here are some suggestions which link to their corresponding OpenStreetMap wiki pages:

- **name=*** (<https://wiki.openstreetmap.org/wiki/Key:name>)
- **addr:housenumber=*** (<https://wiki.openstreetmap.org/wiki/Key:addr>)
- Types of stores:
 - **shop=supermarket** (<https://wiki.openstreetmap.org/wiki/Tag:shop%3Dsupermarket>) for large full service grocery stores.
 - **shop=grocery** (<https://wiki.openstreetmap.org/wiki/Tag:shop%3Dgrocery>) for traditional groceries stores that are not full service.
 - **shop=convenience** (<https://wiki.openstreetmap.org/wiki/Tag:shop%3Dconvenience>) for small local stores.
 - **shop=greengrocer** (<https://wiki.openstreetmap.org/wiki/Tag:shop%3Dgreengrocer>) for stores focused primarily on fruit and vegetables.
- **website=*** (<https://wiki.openstreetmap.org/wiki/Key:website>)
- **opening_hours=*** (https://wiki.openstreetmap.org/wiki/Key:opening_hours)
 - Tips (because the opening_hours tag can be confusing)
 - Use military time.
 - YES: 06:00-22:00
 - NO: 6am to 10pm
 - Use two letter day indicators: Su Mo Tu We Th Fr Sa

- YES: Mo-Fr
- NO: Mon thru Fri
- Do not add extra spaces between two letter day indicators or time ranges
- YES: Mo-Fr 06:00-22:00
- NO: Mo - Fr 06:00 - 22:00
- Separate more complex hours with a semicolon
- YES: Mo-Fr 06:00-20:00; Sa-Su 08:00-16:00
- NO: any other separator besides a semicolon!
- Additional examples:
- 24/7
- Mo-Fr 06:00-22:00; Sa-Su 08:00-16:00
- Mo-Su 06:00-22:00; Th off
- Looking for a helpful interface to help your format this tag properly? Check out **YoHours** (<http://github.pavie.info/yohours/>)

Where to Start

Do you shop at the same grocery stores on a regular basis? Enhance those first. Do you pass by a particularly fancy market that you'd like to shop at regular? Save those details in OpenStreetMap for future retrieval. Maybe you just need a kitchen gadget to get you excited to cook again! What is your go-to store for cooking supplies? Explore the **OpenStreetMap wiki** (https://wiki.openstreetmap.org/wiki/Map_Features) for the appropriate tags to accompany the new features you would like to add to the map!

Interactive Map

Would it be helpful to see where the current grocery related POIs are in OpenStreetMap? There's a map for that! Hover over any of the bright blue highlighted grocery points of interest to bring up an info bubble with links to editing tools that can be used to add additional tags. (While you pan and zoom to your neighborhood, note that stores where you might find groceries only show up once you zoom in to level 16.)



Leaflet | Geocoding by Mapzen

This map is interactive! Open full screen ↗ (<https://mapzen-data.github.io/targeted-editing/te-grocery-stores/map/index.html?grocery>)

Don't know Dublin? Search for your city! Is your go-to grocery store missing all together? Shift-click anywhere on the map to open iD or open your favorite OpenStreetMap editor to start mapping!

If you are trying to decide between creating a point or a polygon to represent your grocery store, a point is great when the store is in a building that contains other businesses and/or residences. If your store occupies the entire building, digitize a building polygon if it doesn't already exist and add your tags to it.

Look for more posts in the **New Year's Resolution** (<https://mapzen.com/tag/new-years-resolutions>) series soon!

Getting Started with OpenStreetMap

Need instructions on how to edit with iD? Here are some links to outstanding tutorials from LearnOSM, the OpenStreetMap wiki, and the United States Department of State's Humanitarian Information Unit:

- **LearnOSM** (<http://learnosm.org/en/>)
- **OSM Beginners' Guide** (http://wiki.openstreetmap.org/wiki/Beginners'_guide)
- **MapGive Learn to Map** (<http://mapgive.state.gov/learn-to-map/>)

Are you a mapping wiz and interested in a more advanced editor? Try out **JOSM** (<https://josm.openstreetmap.de>) with **excellent documentation** (<https://www.mapbox.com/blog/osm-mapping-guide/>) from Mapbox.

Editing with a Mobile Device

With points of interest, I bet you are interested in a mobile app to edit OpenStreetMap while you are on the go.

iOS users can check out **Go Map!!** (<https://appsto.re/us/dafwj.i>) by Bryce Cogswell. This free editor allows you to add features and edit existing features. Check out the **Go Map!!** (http://wiki.openstreetmap.org/wiki/Go_Map!!) wiki page for more details. **Pushpin** (<http://www.pushpinosm.org>) by Fulcrum is another excellent iOS editor for OpenStreetMap points of interest.

Looking for an Android equivalent? **Vespucci** (<https://play.google.com/store/apps/details?id=de.blau.android>) by Marcus Wolschon is a great option. Check out the **Vespucci** (<http://vespucci.io>) home page for documentation and tutorials.

Get outside, increase your local knowledge, and share it with the world through your OpenStreetMap edits!

*Check out all the posts in the **Targeted Editing series** (<https://mapzen.com/tag/targeted-editing>):*

- **Airports** (<https://mapzen.com/blog/targeted-editing-airport-polygons>)
- **Banks** (<https://mapzen.com/blog/new-years-resolutions-money/>)
- **Building Names** (<https://mapzen.com/blog/targeted-editing-name-that-building>)
- **Campus Mapping** (<https://mapzen.com/blog/targeted-editing-campus-mapping/>)
- **Cycleways** (<https://mapzen.com/blog/targeted-editing-cycleways/>)
- **Fitness** (<https://mapzen.com/blog/new-years-resolutions-fitness>)
- **Groceries** (<https://mapzen.com/blog/new-years-resolutions-groceries>)
- **Hospitals** (<https://mapzen.com/blog/targeted-editing-hospital-polygons>)

- **Schools** (<https://mapzen.com/blog/targeted-editing-school-polygons>)
- **Stadiums & Parking** (<https://mapzen.com/blog/targeted-editing-tailgate-mania>)
- **Street Names** (<https://mapzen.com/blog/targeted-editing-no-name-roads>)
- **Transit Colours** (<https://mapzen.com/blog/targeted-editing-transit-colours>)
- **Travel & Lodging** (<https://mapzen.com/blog/new-years-resolutions-travel>)

· 19 January 2016 ·



Indy Hurt

Indy was our resident data scientist lending her geographic expertise to all things "open".

© 2017 Mapzen

WebGL NYC Meetup

We've been pushing the limit of WebGL for a while now with our **Tangram** (<https://mapzen.com/projects/tangram>) map-rendering engine. Binging on open source and open data we've opened entirely new worlds of interactivity, flexibility, control, and let's not forget **Tron** (<https://tangrams.github.io/carousel/?tron>). Who would have thought a map could reach new heights beyond a khaki-colored navigation base?

On Tuesday, Brett Camper and Peter Richardson from our Tangram team get down and dirty with WebGL map display and interactivity at the **NYC WebGL meet-up** (<http://www.meetup.com/NYC-WebGL-Developers/events/227419452/>). If you're doing anything requiring location (and umm... in this day and age, how could you not?) or interested in how to make these new worlds, come! **RSVP** (<http://www.meetup.com/NYC-WebGL-Developers/events/227419452/>)! We want to hear from you.

The deets: Tuesday Jan 26th, 6:30pm | BioDigital Inc, 594 Broadway Suite 1101 | And there's even a map for that (<http://tangrams.github.io/tangram-frame?url=https://gist.githubusercontent.com/meetar/d5433901c32ec7c53226/raw/f74dffafc6febace500bbf64d61c66b3ba150728/scene.yaml&noscroll#18/40.72494/-73.99696>).



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

· 22 January 2016 ·



Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.

© 2017 Mapzen

Shaping Up Your Route Guidance

routing ([/tag/routing](#))

Are you sticking to your 2016 New Year's Resolutions? At Mapzen, our goal was to "shape up" in the new year too, and thanks to some route guidance improvements, the user will be heading in the right (correct) direction. Whether the shape of your path is a fork, a circle, or a 'T', these latest improvements are certain to provide clarity. Since the June **State of the Map US Conference** (<http://stateofthemap.us/2015/quality-route-guidance/>), we have been continuing to improve the narrative with regards to car, bicycle, and pedestrian modes of transportation. We've included some before and after examples to better explain the enhancements.

Car Narrative Improvement

We added code to identify and process forks. Forks typically occur at highway-to-highway transitions with the absence of a ramp or at ramp-to-ramp transitions.



This is a snapshot of the I 95 South/I 895 South highway fork. Now, thanks to the improvement, the user will notice the proper relative direction and interchange information.

Before	After
Continue on I 895 South	<i>Keep right to take exit 62 onto I 895 South toward Annapolis</i>

OpenStreetMap Edit History Export More ▾

39.3369, -76.5164
39.3066, -76.5434
Car (Mapzen) Go

Directions

Distance: 4.1km. Time: 0:03.

- ↑ Head southwest on I 95 South/John F. Kennedy Memorial Highway. 1400m
- ↑ Keep right to take exit 62 onto I 895 South/Harbor Tunnel Thruway toward Annapolis. 2.7km
- Your destination is on the left.

Directions courtesy of [Mapzen](#)

Keep right to take exit 62 onto I 895 South/Harbor Tunnel Thruway toward Annapolis.

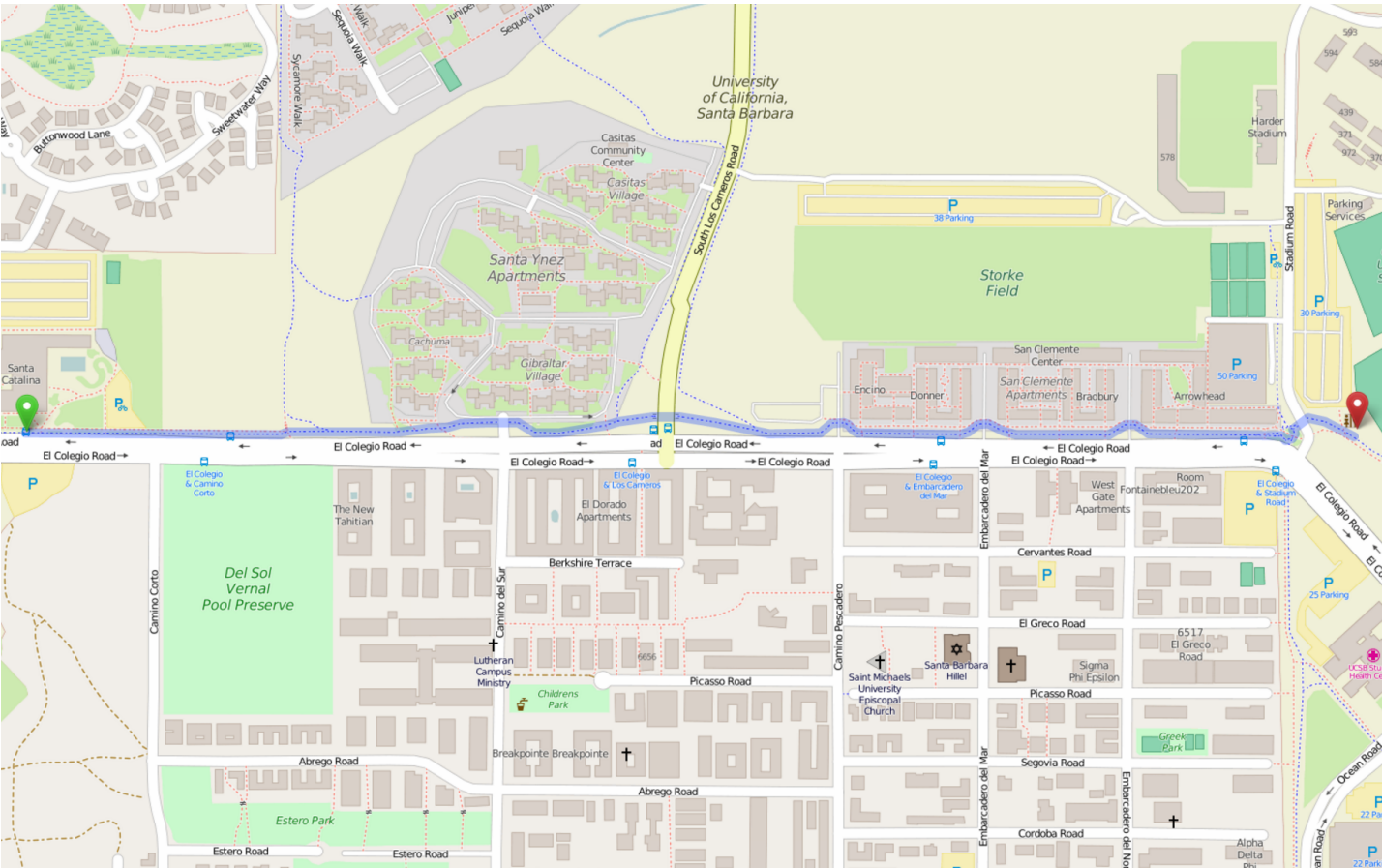
© OpenStreetMap contributors ♥ Make a Donation

(http://www.openstreetmap.org/directions?engine=mapzen_car&route=39.3369%2C-76.5164%3B39.3066%2C-76.5434)

View on **OpenStreetMap** (http://www.openstreetmap.org/directions?engine=mapzen_car&route=39.3369%2C-76.5164%3B39.3066%2C-76.5434)

Bicycle Narrative Improvement

The bicycle narrative needed to shed unwanted pounds maneuvers and properly handle the cycleway circles (roundabouts). We collapsed the unnamed maneuvers and added the cycleway label. Also, we fixed the exit spoke count for cycleway circles.



(http://www.openstreetmap.org/directions?engine=mapzen_bicycle&route=34.41756%2C-119.86799%3B34.41758%2C-119.85277#map=17/34.41759/-119.86039)

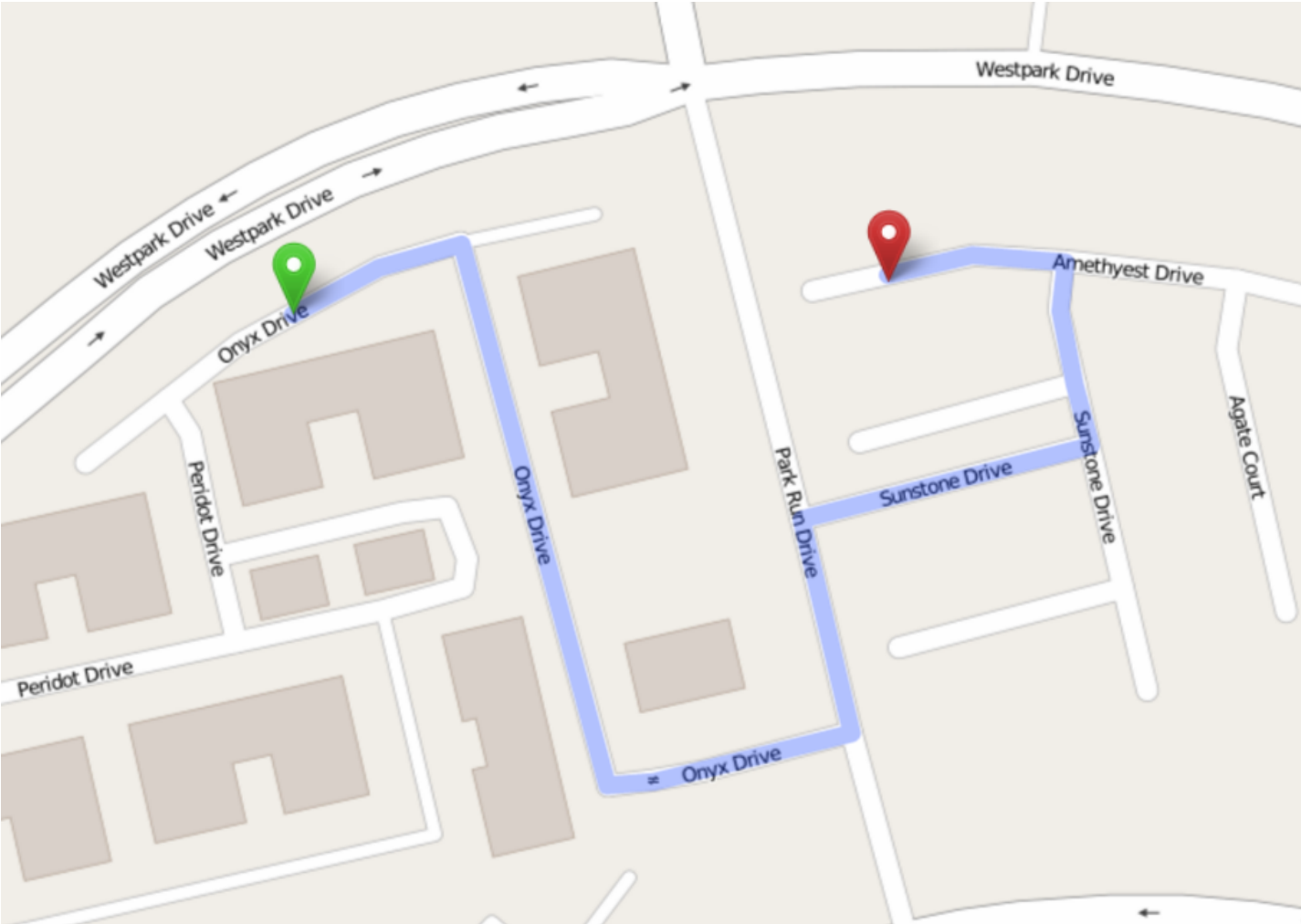
Before	After
Go east.	Head east on cycleway.
Continue. Continue. Continue. Continue. Continue. Bear right. Continue. Continue. Bear right. Bear right. Continue. Turn right. Bear left. Continue. Bear left. Bear right. Bear left. Continue. Bear left. Bear right. Continue. Continue. Continue.	(Collapsed unnamed maneuvers)
Enter the roundabout and take the 1st exit.	Enter the roundabout and take the 2nd exit.
Exit the roundabout.	Exit the roundabout onto cycleway.
Turn right. Continue. Turn right.	(Collapsed unnamed maneuvers)

You have arrived at your destination.	<i>Your destination is on the left.</i>
---------------------------------------	---

View on **OpenStreetMap** (http://www.openstreetmap.org/directions?engine=mapzen_bicycle&route=34.41756%2C-119.86799%3B34.41758%2C-119.85277#map=17/34.41759/-119.86039)

Pedestrian Narrative Improvement

We updated the logic to call out a turn at a 'T' intersection even if the street names are the same, making it clear to the user which way to go at the 'T' (Onyx Drive in the example below). We softened the start maneuver (Go vs. Head) and added the side of street to the the destination maneuver.



(http://www.openstreetmap.org/directions?engine=mapzen_foot&route=38.92828%2C-77.22812%3B38.92849%2C-77.22634)

Before	After
Go northeast on Onyx Drive.	<i>Head northeast on Onyx Drive.</i>

	<i>Turn right to stay on Onyx Drive.</i>
Turn left onto Park Run Drive.	<i>Turn left onto Park Run Drive.</i>
Turn right onto Sunstone Drive.	<i>Turn right onto Sunstone Drive.</i>
	<i>Turn left to stay on Sunstone Drive.</i>
Turn left onto Amethyst Drive.	<i>Turn left onto Amethyst Drive.</i>
You have arrived at your destination.	<i>Your destination is on the right.</i>

View on **OpenStreetMap** (http://www.openstreetmap.org/directions?engine=mapzen_foot&route=38.92828%2C-77.22812%3B38.92849%2C-77.22634)

We hope these make it even easier to integrate Mapzen Turn-by-Turn into your apps and maps. Check out the **documentation** (<https://mapzen.com/documentation/turn-by-turn/>) and **let us know if you have any questions** (<https://twitter.com/intent/tweet?text=@Mapzen%20@ValhallaRouting%20Hi!>).

· 25 January 2016 ·



Duane Gearhart

Duane is a software engineer @mapzen specializing in quality route guidance and real-world route analysis.

© 2017 Mapzen

Station Relations

data (/tag/data) **osm** (/tag/osm) **vector-tiles** (/tag/vector-tiles)

Having stations on the map seems obvious; stations are important both for someone travelling by public transit and as landmarks for anyone driving. Many are prominent, ornate buildings occupying a large footprint in the urban landscape, but others can be small, spread-out, tucked into a corner or buried underground. When looking at an overview of a city, it may not be possible or desirable to show *all* the stations, and it can be hard to tell: Which stations are important? It seems like it would be useful to be able to compare stations and rank them by some measure of “importance”.

What’s a useful metric for importance of stations? The reason we want to show them on the map is that the more important stations are more likely to be known by a larger number of people, due to having travelled via them. The ideal metric would, therefore, be the number of passengers who transit via the station in a given period of time.

Unfortunately, the availability of passenger number data is patchy, and collecting it globally would be a lot of work. Thankfully, there are people taking on the difficult task of collecting **open data about transit** (<https://transit.land/>) and **places, generally** (<https://whosonfirst.mapzen.com/>). At the moment, neither of these contain passenger numbers for every station in the world, so it looks like we’ll need something else to fall back on.

If we have access to the structure of the transit network, but nothing else, can we deduce anything useful from that? It turns out that this is called **Network Theory** (https://en.wikipedia.org/wiki/Network_theory). One example of network theory is Google’s **PageRank** (<https://en.wikipedia.org/wiki/PageRank>) algorithm, which can be used to generate better search results by looking at the number of web sites which link to a given page; pages with more inbound links are likely to be better results. By analogy, the more important stations will be the ones bringing more passengers in by being on more train or subway routes. Additionally, stations part of a large number of *different* routes are more likely to be places where people change between routes, and therefore more likely to be well-known.

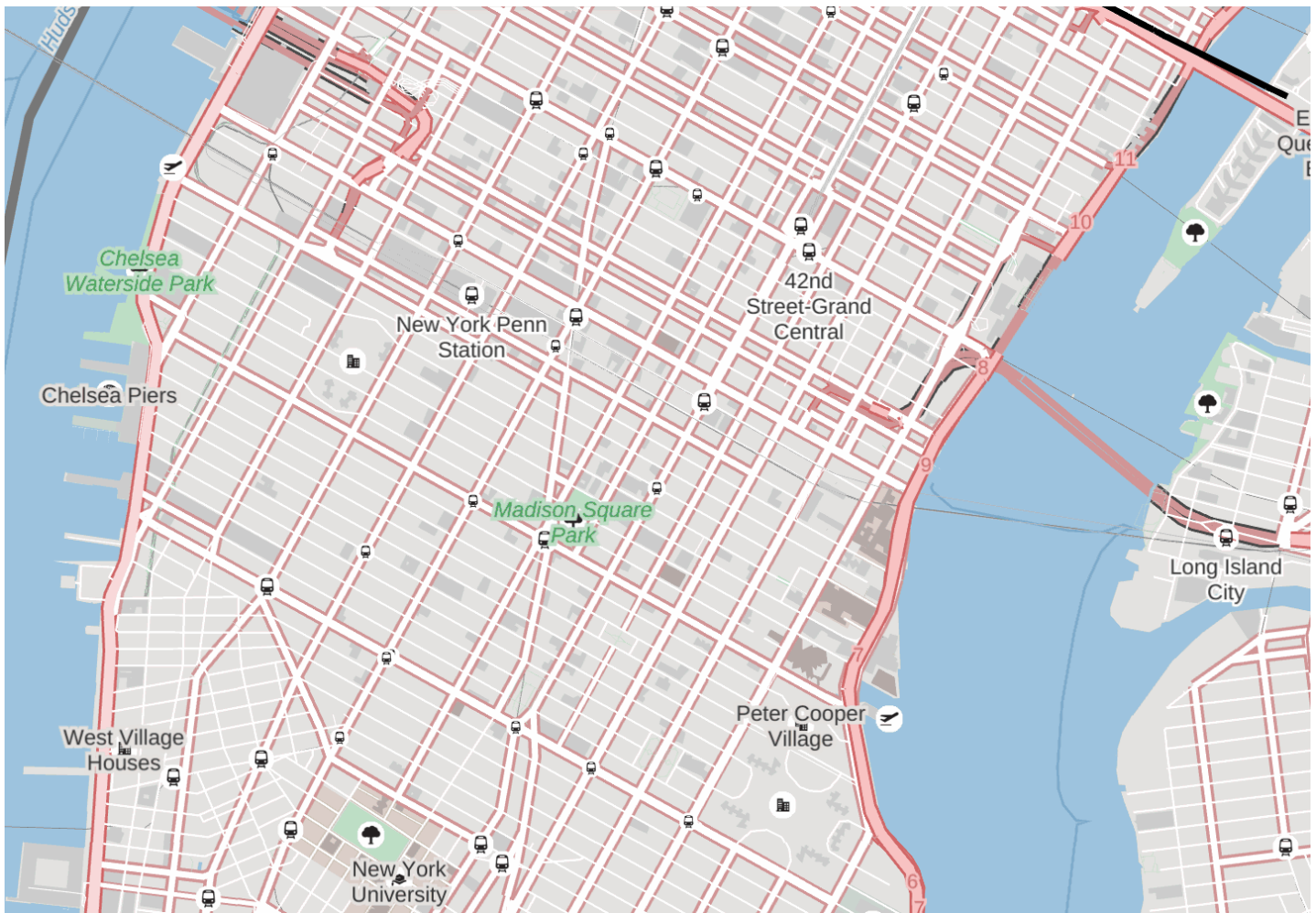
Some places in the world have many train stations, and in others trains are relatively rare. Also, some places in the world have more of this data mapped. This means that whatever metric we can derive from the structure of the transit network is likely to be valid only for comparing

between stations within roughly the same area. Trying to compare globally is likely to run into these issues, but if we're only looking to use this metric to rank stations which will be visible on the map at the same time, then we should be okay.

In summary, we might be able to approximate the relative importance of nearby stations by counting the number of different *routes* that they're associated with.

It's worth noting that *route* can have different meanings in different contexts, and is especially overloaded when it comes to public transit. In this post, I'll be trying to use *route* to mean a named line (physical track) or regular service. This runs against some **good advice** (<http://humantransit.org/2011/02/watching-our-words-route-or-line.html>) to use *line* for things where the service runs "so often that you don't need a schedule". Using *route* for this, although undesirable from that point of view, fits better with the source data, which will be explained in the next section.

To see what a difference it can make to show only the more important stations, have a look at this animation showing the stations with fewer routes fading in and out. Penn Station, Times Square, Grand Central and Long Island City all remain, as the stations serving the largest number of routes. On this map, which doesn't have many other labels, it might seem like having more is better, but notice how the stations crowd out other features such as New York University and Madison Square Park - in this case, fewer is better.



Diving into the data

If we want open data about transit networks for the whole world, there's really only one place to get it: **OpenStreetMap (<http://openstreetmap.org>)** (OSM), the “wiki world map” is maintained by thousands of volunteers world-wide who put a huge amount of effort into mapping things, including transit networks. However, stations in OSM are often just points (called “nodes” in OSM jargon) with little extra information included as part of the object.

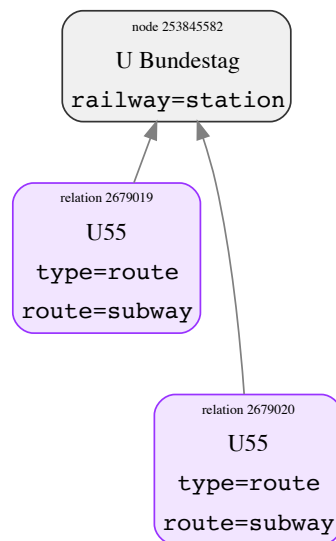
OSM doesn't, and nor should it, contain ephemeral and non-verifiable data such as schedules. This doesn't mean that schedules aren't important - they are! For schedule data, we would have to rely on openly published “feed” information, which can be found at **Transitland (<https://transit.land/>)**. For accurate trip planning it is necessary to know both the schedule data and the geographic data, which makes the answers time-dependent. This is exactly what makes a trip planner useful! However, the metric of “importance” for a given station probably shouldn't be time-dependent for a general-purpose map (it sounds like an intriguing idea for a transit-

specific map, though - take a look at this **service frequency-dependent map** (<https://mapzen.com/blog/the-transit-dimension-transit-land-schedule-api#frequency-is-freedom:-an-example>!)), so we can use the geographic data in the OSM data.

OSM's data model is more sophisticated than nodes alone, and also includes “ways” which join up nodes to form lines or polygons, and “relations” which describe relationships between nodes, ways and other relations.

Relations are powerful, and sometimes confusing, objects. They exist to provide a flexible way to introduce structure to the OSM data without having to impose a rigid, limited set of permitted structures from outside (called an “**ontology**” (https://en.wikipedia.org/wiki/Ontology_%28information_science%29)).

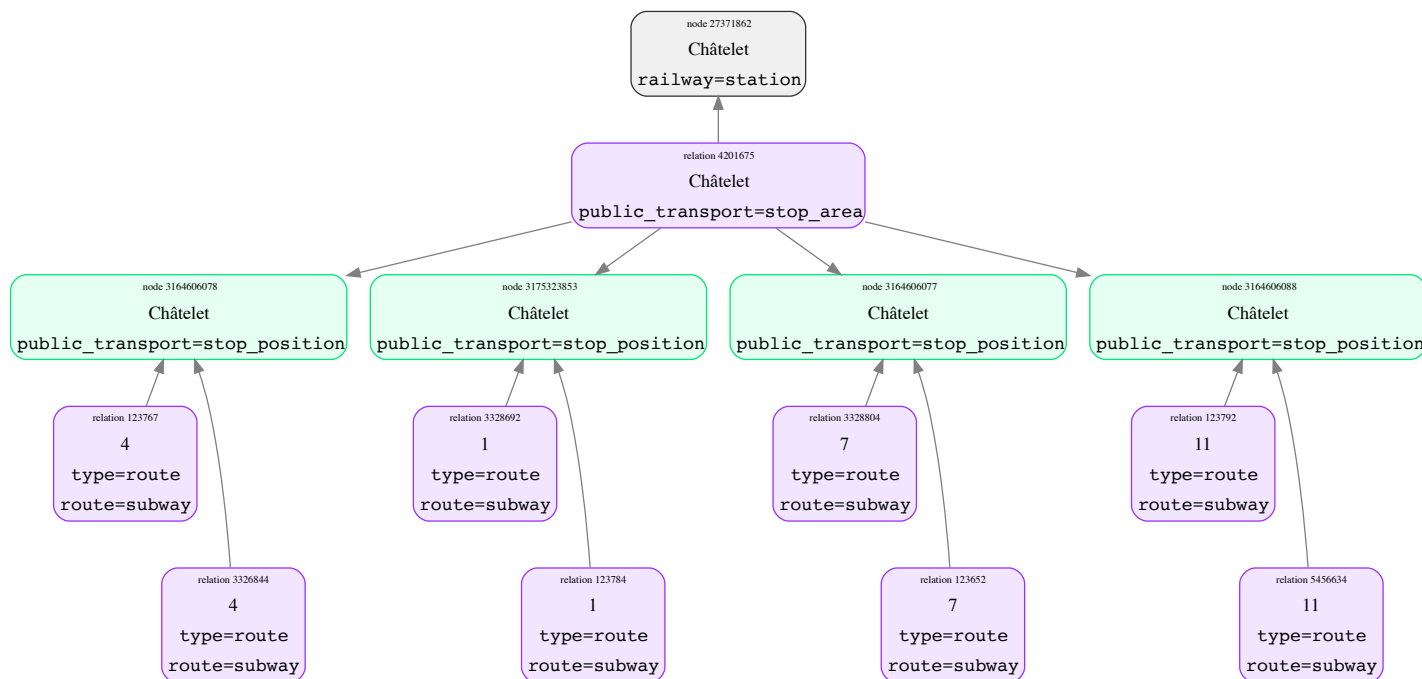
Let's take a look at an example of a station node - **U Bundestag** (<https://www.openstreetmap.org/node/253845582>) in Berlin, Germany. We can see that it is part of a **route relation** (<https://www.openstreetmap.org/relation/2679020>) which tells us that the U-bahn line U55 goes through U Bundestag.



This is great! But sadly not all stations are directly members of route relations, and many stations are composed of different collections of platforms or different kinds of lines. It is necessary to dig a bit deeper to get all the information.

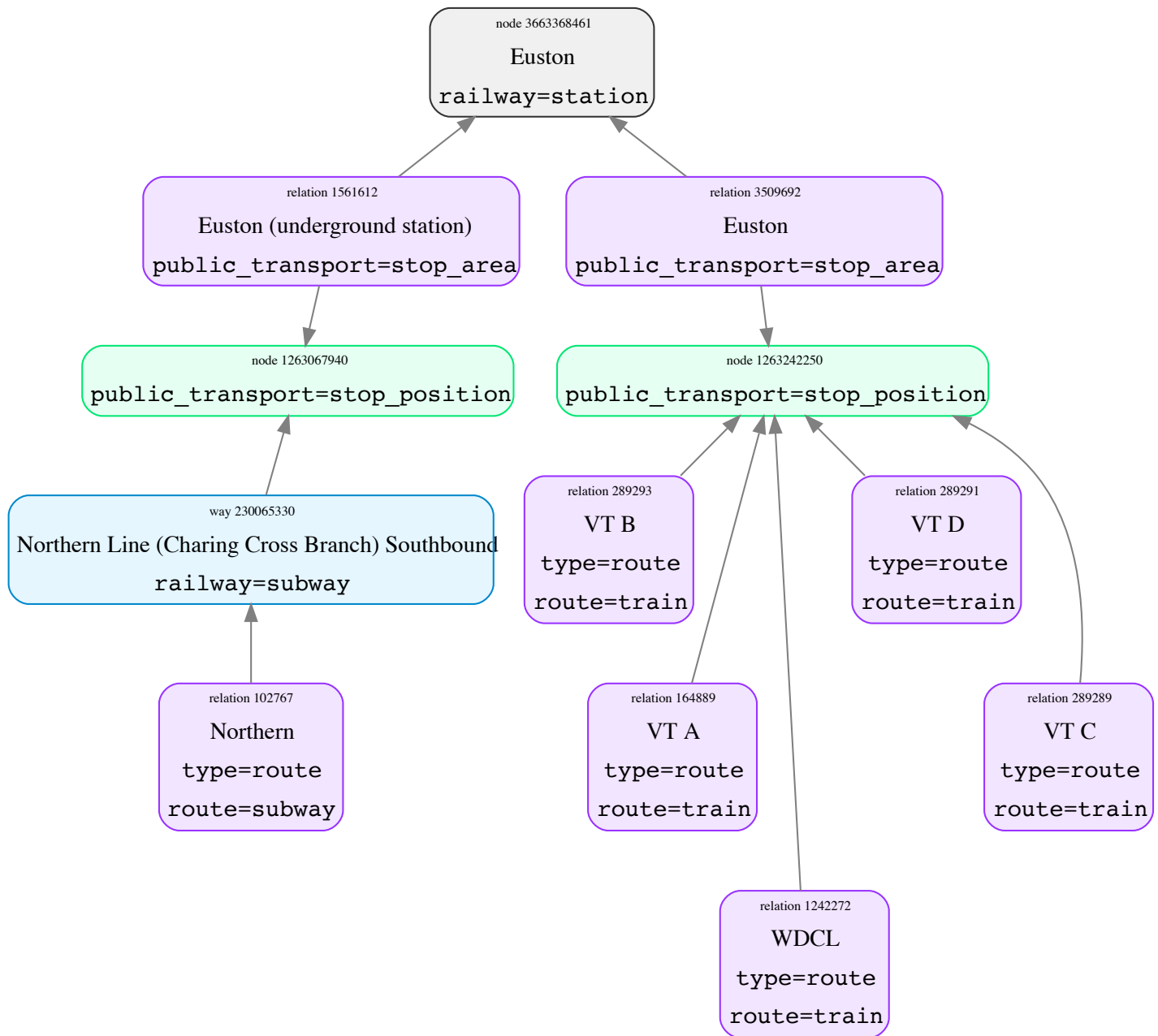
It's worth noting that this “route” relation is, in all likelihood, a *line* in the sense of **good transit terminology** (<http://humantransit.org/2011/02/watching-our-words-route-or-line.html>). In OSM, the relations representing either named physical lines or named regular services are called “routes”, and since this article is mainly about navigating the data in OSM, using the not-so-good transit terminology would seem best for clarity.

Let's take a look at a more complex example of a station node, **Châtelet** (<http://www.openstreetmap.org/node/27371862>) on the Paris Métro, France.



We can see that the station node is included in a relation which has data saying it is a **stop area** (<http://www.openstreetmap.org/relation/4201675>). Stop area relations link several stops together, and indicate that they're grouped as a single unit intended for changing between lines or modes of transit. Stop area relations generally contain several nodes with a data **identifying them as stops** (<http://www.openstreetmap.org/node/3164606078>) or stations in their own right.

Stations and stops are often directly included as members of route relations, as in the case of Châtelet, but sometimes are not. When they're not, we can look to see if the node is used as part of a way describing the geometry of the railway track which, itself, might be included as a member of some route relations. Let's take a look at **Euston station** (<https://www.openstreetmap.org/node/3663368461>) as an example.



The Euston station node is part of two stop area relations, one of which is the **underground part of the station** (<https://www.openstreetmap.org/relation/1561612>). This stop relation contains a node for the **stop position** (<https://www.openstreetmap.org/node/1263067940>) on the **railway line** (<https://www.openstreetmap.org/way/230065330>) for the Northern line **route relation** (<https://www.openstreetmap.org/relation/102767>). Whew! That was a lot of indirection to go through!

We've now seen several different ways of navigating the OSM data structures to get from the original station node to many route relations, each of which is potentially a different route that a passenger could travel on. Taking all of these routes, we can estimate how much of a "hub" this particular station is, and use that as a proxy for how visually important it should be.

Extra information

Much of this is described for many different forms of public transit on the OSM **wiki** (http://wiki.openstreetmap.org/wiki/Public_transport). However, when using data from OSM it's important to use real examples and test things on real data from the **OSM data dumps** (<http://planet.openstreetmap.org/>) or **metro area extracts** (<https://mapzen.com/data/metro-extracts>). What really matters is that the code works, and calculates the data you're interested in. Documentation on the wiki can be helpful, but is often out of date with respect to the data - and sometimes reflects an ideal of what the data *could* be, rather than what it *is*. If there's ever a conflict between what the wiki says and what real data exists, then the real data is right!

Implementing it

We've implemented this on top of a rendering database created with `osm2pgsql` (<https://github.com/openstreetmap/osm2pgsql>) as part of our **vector-datasource** (<https://github.com/mapzen/vector-datasource/>) tile rendering. Before we jump into **the code** (<https://github.com/mapzen/vector-datasource/blob/8ca608cc9a9e61d4d5e97146df955daf3518b81b/data/functions.sql#L627-L711>), which is far from straightforward, it's worth going through how `osm2pgsql` stores information about relations in the database.

The output tables which `osm2pgsql` creates by default contain points, lines and polygons processed from OSM data, but no information about relationships between OSM objects. To get to that information, we have to look beyond the output tables and into the guts of `osm2pgsql` ; the "middle" tables.

The middle tables exist because `osm2pgsql` is able to process **updates from OSM** (<http://wiki.openstreetmap.org/wiki/Planet.osm/diffs>) and needs to keep track of the current state of all items to know what has been updated and fetch dependencies of objects. We can make use of the middle tables to fetch information about relations.

Relations are stored in the middle tables in the following schema:

```
CREATE TABLE planet_osm_rels (
  id bigint NOT NULL,
  way_off smallint,
  rel_off smallint,
  parts bigint[],
  members text[],
  tags text[]
);
```

Each row in `planet_osm_rels` stores a single relation, identified by ID `id`. The `parts` are the IDs of all of the members of the relation, stored in an array with the nodes first, then the ways, then the relations. The index of the first way is given by `way_off`, and the first relation by `rel_off`. This is a compact method of storing the IDs, but leads to a problem when we want to join this table to something else: **GIN indexes**

(<http://www.postgresql.org/docs/current/static/gin-intro.html>) can only help us identify rows with a particular ID anywhere in the array, not within specific sections. So we have to do a secondary filter to ensure we don't pick up anything other than the relationships we are looking for. For example, when we are looking for relations which contain a node with ID `station_node_id`, we need a filter like this:

```
r.parts && ARRAY[station_node_id]
AND r.parts[1:r.way_off] && ARRAY[station_node_id]
```

The first part allows **PostgreSQL** (<http://www.postgresql.org/>) to use the GIN index. The second line ensures that we only pick up relations containing that node, not any relations which might contain a way with the same ID.

We also only want to search for specific types of nodes, ways and relations at various points in the algorithm, so we filter on their `tags`, which are key-value pairs of strings. For relations in the `planet_osm_rels` table, these are encoded as a flat list, for example: `[key1, value1, key2, value2]`, and so we have a function `mz_rel_get_tag` (<https://github.com/mapzen/vector-datasource/blob/8ca608cc9a9e61d4d5e97146df955daf3518b81b/data/functions.sql#L451-L482>) to decode that list and fetch the value for a given key.

The algorithm proceeds with four main steps:

1. We find stop area relations containing the original station node (or way).
2. We find any station or stop nodes contained in these stop area relations.

3. We find any railway lines which make use of any of the station or stop nodes.
4. We find any route relations which contain any of the station or stop nodes or any of the railway lines.

The full code for this **function** (<https://github.com/mapzen/vector-datasource/blob/8ca608cc9a9e61d4d5e97146df955daf3518b81b/data/functions.sql#L627-L711>) can be found in our **vector-datasource** (<https://github.com/mapzen/vector-datasource/>) repository on Github.

Stop Area Groups

One thing that hasn't been mentioned yet is another layer of indirection: **stop area groups** (http://wiki.openstreetmap.org/wiki/Relation:public_transport). These are infrequently used to group together multiple stop areas. Adding another layer of indirection would be easy, but might require us to think slightly differently about how to aggregate multiple stations which might be part of the same stop area group.

Grouping and aggregating stations at lower detail levels would be very nice, and immediately seems desirable, but can lead to unexpected visual artefacts: When a station dis-aggregates into multiple parts, what had been a very important station-group might fracture into many individual stations, each of which is not important enough to display. It would appear as if the station simply disappeared, which isn't good.

The Shortcut

If that all sounds great, but a little bit complicated, then you might be interested in our **vector tile service** (<https://mapzen.com/projects/vector-tiles>) and our fully-featured tile styles such as **Cinnabar** (<https://mapzen.com/blog/introducing-refill-cinnabar-and-zinc-styles-for-tangram>). These already have the logic above baked-in and ready to use! Just look for the `kind_tile_rank` attribute on `kind: station` features. This starts at 1 for the most important and increases as the features of that kind get less important in a given tile. In conjunction with **Tangram** (<https://mapzen.com/projects/tangram>)'s powerful and flexible styling rules, this gives you fine control over how best to display your map to your users.

Ready to get started? **Sign up for an API key** (<https://mapzen.com/developers>) today!

Mapping this data

If your important local stations aren't being displayed at mid ranges on Mapzen maps, it might be because we aren't able to find the links between the station node or way and the route relations. You can create relations for each route, where a route is usually a named line such as “**Northeast Corridor** (https://en.wikipedia.org/wiki/Northeast_Corridor)” or a named service such as “**California Zephyr** (https://en.wikipedia.org/wiki/California_Zephyr)”. You can either include the stations directly as part of the route relation, or create a **stop area relation** (http://wiki.openstreetmap.org/wiki/Tag:public_transport%3Dstop_area) containing the station node or way and any stop nodes which are part of the route.

By adding stop area and route data, it not only helps us figure out which stations might be more important for display on the map, but also helps anyone else who's looking to build trip-planning or transit-related applications on top of OpenStreetMap. Thank you in advance!

· 27 January 2016 ·



Matt Amos

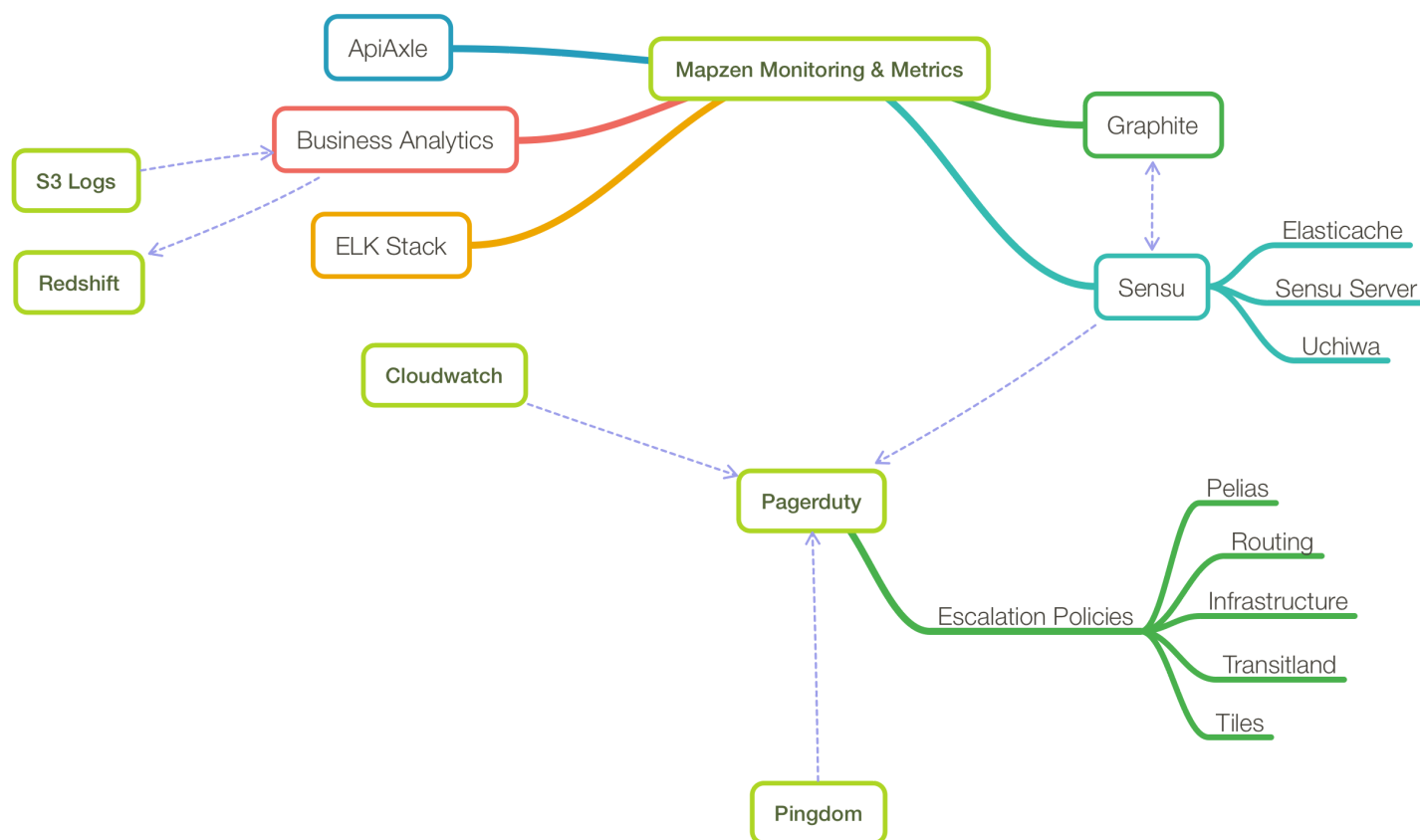
OpenStreetMap developer-contributor and chilli enthusiast. Writes vector tile and other data-mastication tools at Mapzen.

© 2017 Mapzen

Monitoring & Metrics Infrastructure

engineering (/tag/engineering)

As your infrastructure grows, you need to know how it's operating. The second post **in our Engineering at Mapzen series** (<https://mapzen.com/tag/engineering>) gives you a little taste of how we do that here.



Monitoring

A good monitoring solution is, in my estimation, first and foremost flexible. At Mapzen we have a mix of development teams: those who essentially run and manage their own operational deployments, as well as those who rely more heavily on a small operations team, and everything in between. We need to be able to tune any solution to suite a specific team's needs. Here's how we do it...

Sensu

Sensu (<https://github.com/sensu>) is an incredibly versatile monitoring solution that we rely on heavily. We've been running it in production almost from the first day Mapzen opened shop (I'll take this moment to brag about being employee number two). Sensu has a number of things going for it that led to its selection:

- easy integration with a number of services such as Pagerduty for alert notifications
- a large community that has produced a wealth of code that can extend the system
- easily extensible, it doesn't force you to write ad hoc monitoring code in any particular language: use ruby, python, bash or anything else your systems support
- it uses a queuing system as the backend for checks, which means we can leverage **Elasticache** (<https://aws.amazon.com/elasticache/>) with a Redis backend, and easily scale the systems pulling events off the queue
- it has a simple API that allows us to interact with events (silence, acknowledge or resolve), schedule maintenance to reduce unnecessary alerts, and to easily add and remove systems

Here's a contrived example of a bash script executing a sensu compatible check. It simply counts the files in /tmp and exits with appropriate exit status if either the warning or critical thresholds are breached. The sky's the limit, simply exit your checks with the appropriate status codes to trigger the desired alert status.

```
#!/bin/bash

warn_thresh=10
crit_thresh=20
tmpfiles=$(ls -l /tmp/ | wc -l)

if [ ${tmpfiles} -gt ${crit_thresh} ]; then
    echo "Critical: number of files exceeds crit threshold!"
    exit 2
elif [ ${tmpfiles} -gt ${warn_thresh} ]; then
    echo "Warning: number of files exceeds warn threshold!"
    exit 1
else
    echo "OK"
    exit 0
fi
```

We also run some simple chef code, making use of the **sensu-chef** (<https://github.com/sensu/sensu-chef>) cookbook, to add and remove systems as they come online or are removed from service:

Add a client:

```
sensu_client node[:mapzen_sensu_clients][:sensu_client_name] do
  address      node[:ipaddress]
  subscriptions node[:mapzen_sensu_clients][:subscriptions_array]
  additional(
    keepalive: {
      thresholds: { warning: 180, critical: 300 },
      handlers: node[:mapzen_sensu_clients][:handlers_array],
      refresh: 1800,
      handle: keepalive
    }
  )
end
```

Remove a client:

```
http_request 'sensu-remove-client' do
  action      :delete
  url         "http://#{user}:#{secret}@#{node[:sensu][:api][:host]}:#{node[:sensu][:api]
  retries     3
  retry_delay 10
end
```

Pingdom

Pingdom (<https://www.pingdom.com/>) is a third party service that allows you to run checks on services from all over the world. They're an old staple, so I won't go into any detail here. Suffice to say we like the service, but the interface could use some help. If you're up on any comparable solutions, I'd love to hear your suggestions.

Pagerduty

Pagerduty (<https://www.pagerduty.com>) is another third party service that we leverage heavily. Our company structure is such that we have different teams working on very specific pieces of the mapping stack. Obviously, it isn't desirable for the Search team to get woken up at 2am for something that's broken in the Routing stack. Pagerduty's services, along with Sensu's flexibility, allow us to ensure that the right people get notified. Pagerduty is the funnel through which every alert we generate flows. Keeping all our alert traffic centralized makes it easier to manage and track.

Metrics

ELK Stack

The **Elastic ELK** (<https://www.elastic.co/products/logstash>) stack probably doesn't need much of an introduction. Composed of an Elastic backend, Logstash as a log parser/shipper and Kibana as a front end for visualization, we use it heavily to keep ongoing tabs on system health over long periods of time.

Here's a quick example:

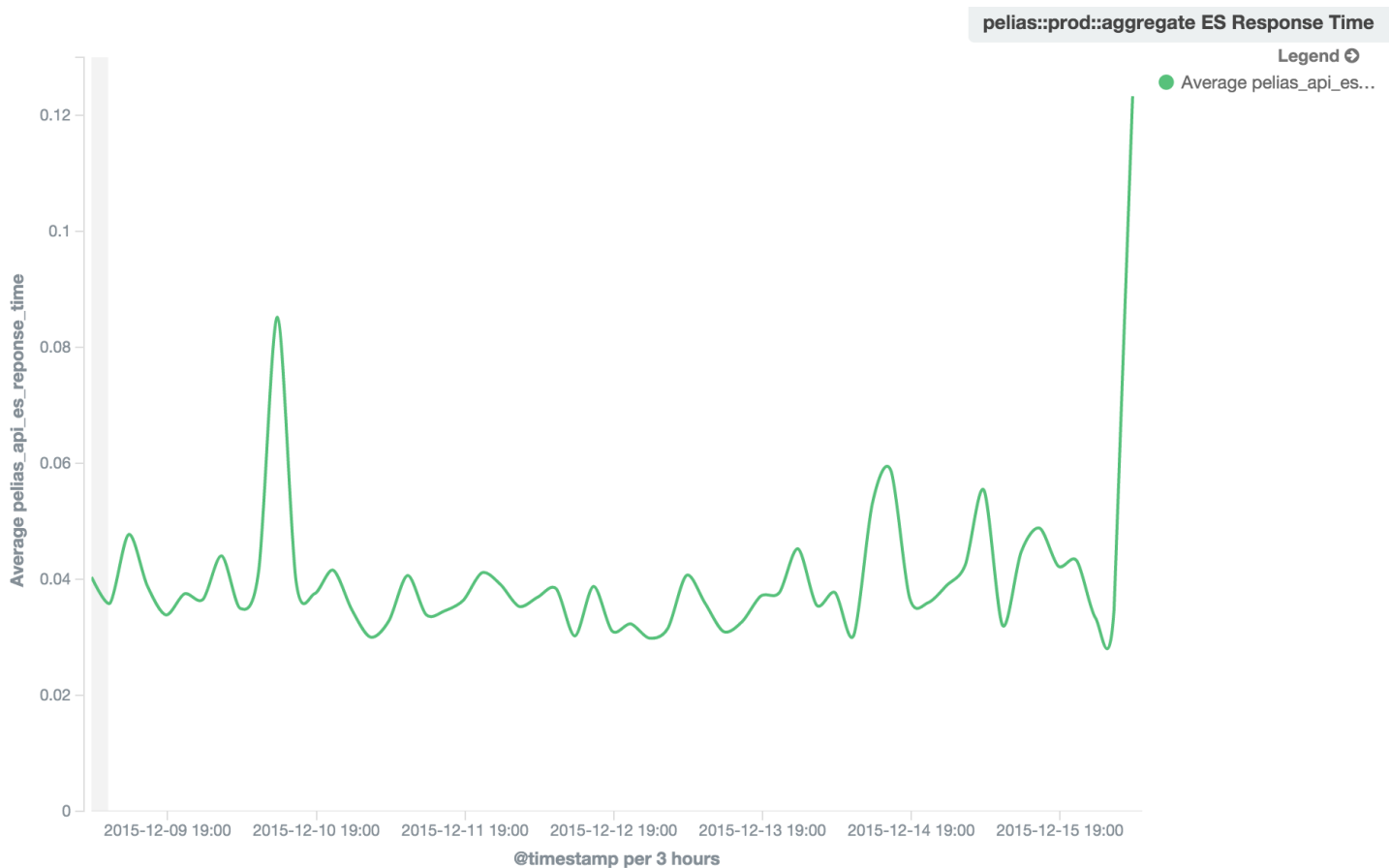
```
input {
  file {
    type => "pelias-api-logs"
    path => [ "/var/log/pelias-api/current" ]
    tags => [ "<%= node[:mapzen][:environment] %>", "<%= node[:hostname] %>" ]
  }
}

filter {
  # get took_millis from slowlog
  if [type] == "pelias-api-logs" {
    grok {
      match    => { "message" => "%{GREEDYDATA} time elasticsearch reported: %{NUMBER:pelias-api-es-response-time}" }
      add_tag => [ "pelias-api-es-response-time" ]
    }
  }
}

output {
  zeromq {
    address    => ["tcp://<%= node[:mapzen][:logstash][:host] %>:2120"]
    topology => "pushpull"
  }
}
```

We run a centralized ELK stack for all our systems, and as you might notice from this client configuration snippet, we use the zeromq output with a couple dedicated logstash processes to service incoming traffic.

This particular config, used for Mapzen Search, is pulling response time data from the API log and shipping it off. Using this data, we can generate a graph like the one you see below, which helps us track response time performance:



Graphite/StatsD

Another system that probably needs no introduction, **Graphite** (<http://graphite.wikidot.com/>) is a bit long in the tooth now, but still a useful tool. Specifically, we use Graphite in conjunction with Sensu to generate alerts off of specific data points. For example, OSM planet update times are shipped off to graphite. We then watch that metric as part of a Sensu check and make sure the time doesn't rise above a given threshold, after which we generate an alert (or in other cases perform some remediation without ever sending an alert).

Cloudwatch

If it runs on AWS, there's a **Cloudwatch** (<https://aws.amazon.com/cloudwatch/>) metric for it. Again, we ship any alerts generated here off to Pagerduty.

Closing Remarks

You may have noticed, if you looked closely at the infrastructure diagram up top, that I've left a few things off, namely the Business Analytics system and ApiAxle. I'll leave these as topics for future blog posts. In the meantime, if you're interested in more detail of the monitoring/metrics system I've outlined, get in touch!

*If you liked this, check out the first post in the **Engineering at Mapzen series** (<https://mapzen.com/blog/engineering-series>) about our **Search data pipeline and ElasticSearch** (<https://mapzen.com/blog/mapzen-search-data-pipeline>).*

· 02 February 2016 ·



Grant Heffernan

Grant is a sysadmin with fingers in all of Mapzen's pies. Egli non si senta italiano, ma per fortuna o purtroppo...

© 2017 Mapzen

Targeted Editing – Tailgate Mania

osm (/tag/osm) **targeted-editing** (/tag/targeted-editing)

Maps are current as of Oct 2016.

When parking lots become celebration destinations...

Welcome to our **Targeted Editing** series where today you can cheer on your favorite sports team, or make **OpenStreetMap** (<http://www.openstreetmap.org>) data supreme! (Both take about the same amount of time.)

Here in San Francisco, **Super Bowl City**

(<https://whosonfirst.mapzen.com/spelunker/id/420561633/>) is attracting football fans from far and wide. Firing up a barbecue grill in a stadium parking lot as part of a tailgate party is an American pastime. Whether you are here in the Bay Area for the game, or you have plans to attend games at other stadiums this year, you are going to appreciate being able to find the parking areas on a map!

In North America, there are well over 3,000 **stadiums**

(<https://wiki.openstreetmap.org/wiki/Tag:leisure%3Dstadium>) in the OpenStreetMap data. Nearly half of them have parking areas nearby, but there are still lots of opportunities to add parking aisles and connect them to the road network. Access points for parking areas add a lot of value, too.

Read on for some basic uses for parking features, and where you can check to see if you can use your **local knowlege** to add some of these features where they are needed.

How are parking areas, parking aisles, and access points used?

1. **Help with search.** Finding parking is often a big part of trip planning.
2. **Help with routing.** While your final destination may be a point of interest, you are likely driving to that location's associated parking area. When parking areas also have an access point, it can be a great place to set as a destination because drivers know what to do once

they arrive at a parking lot. In the absence of access points, parking aisles connected to the road network can be used for similar purposes.

3. **Help with map display.** Parking lots are valuable features that drivers rely on, and parking aisles look beautiful on a map. They provide helpful visual cues that are part of the wayfinding process.

We are missing some parking features here!

Check out this table of some metro areas that have hosted Super Bowl games over the years:

Metro Area	Stadiums	with Nearby Parking	with Nearby Parking Aisles
Arlington, TX	5	60%	40%
Atlanta, GA	25	48%	32%
Detroit, MI	25	68%	20%
Houston, TX	42	79%	40%
Indianapolis, IN	10	70%	70%
Jacksonville, FL	6	100%	83%
Los Angeles, CA	46	61%	39%
Miami, FL	25	48%	32%
Minneapolis, MN	25	72%	52%
New Orleans, LA	8	25%	13%
Phoenix, AZ	26	88%	58%
San Diego, CA	8	50%	38%
San Francisco Bay Area, CA	43	63%	30%
Tampa, FL	15	93%	53%

Sort the columns by clicking the headers. The **count of stadiums** (https://github.com/mapzen-data/targeted-editing/blob/gh-pages/queries/stadium_parking.sql#L2-L3) includes all stadiums greater than 2,000 square meters so we can expand our evaluation beyond football, and the stadiums are checked for parking areas and parking aisles within 100 meters.

Most of the stadiums in the Florida cities had parking areas nearby! The numbers start to drop off a bit when we start checking for parking aisles. It may seem like the OpenStreetMap data for parking areas and parking aisles near stadiums is lacking quite a bit in some important markets, but there is more to consider. The search tolerance of 100 meters will be too small in some areas. Additionally, the lack of parking aisles in parking areas tagged as parking garages is not uncommon.

The data comes from our **metro extracts** (<https://mapzen.com/data/metro-extracts/>) and some of the extracts are very large. For instance, the metro extract for Detroit extends north beyond Flint, and west beyond Ann Arbor! The Los Angeles and San Francisco Bay metro extracts are equally enormous. Bounding boxes can be used to trim down the analysis, but when so many stadiums are just outside of their namesake city limits, larger areas are a welcome option.

How you can help improve stadium parking:

Here is a map highlighting stadiums. Hover over one of the bright blue highlighted stadium icons or a parking area to bring up an info bubble with links to editing tools that can be used to add features. We have made the existing parking areas a light purple on the map so it is easier to see where parking aisles **might** be missing. Remember, some of the parking areas might be parking structures.



Leaflet | Geocoding by Mapzen

This map is interactive! Open full screen ↗ (<https://mapzen-data.github.io/targeted-editing/te-stadium-parking/map/index.html?stadiums>)

(While you pan and zoom to your neighborhood, note that parking areas only show up once you zoom in to level 15.)

See the wiki pages for **amenity=parking**

(<https://wiki.openstreetmap.org/wiki/Tag:amenity%3Dparking>), **service=parking_aisle**

(http://wiki.openstreetmap.org/wiki/Tag:service%3Dparking_aisle), **oneway**

(<http://wiki.openstreetmap.org/wiki/Key:oneway>), and **access=customers**

(<https://wiki.openstreetmap.org/wiki/Tag:access%3Dcustomers>).

- **oneway=yes** can be added to parking aisles when the flow of traffic is restricted to one direction. The direction will be the direction in which the parking aisle is digitized.
- For existing parking aisles, if the direction is incorrect, use the “reverse way” tool in iD to reverse the direction of the line to match the flow of oneway traffic. This may also require adding additional nodes to the way to ensure that tags are applied to the appropriate segments.
- Make sure parking aisles are connected to the road network in the appropriate locations. To be **connected** (http://wiki.openstreetmap.org/wiki/Unconnected_ways), two roads (or a road and a parking aisle) must share a node.
- If you know where the parking entrance is located, add a node at that location, make sure it is also connected to the road network, and tag it with **access=customers**.
- Last but not least, including your sources with the **source** (<https://wiki.openstreetmap.org/wiki/Key:source>) key is incredibly helpful. This includes the imagery you used to trace features, and any authoritative websites as long as they allow their use for this purpose.
- Not familiar with the San Francisco Bay Area? Search or pan over to your home town to contribute your local knowledge to the map. You will see your changes right away in OpenStreetMap. You will also be able to see them in all versions of the **Mapzen Vector Tiles** (<https://mapzen.com/projects/vector-tiles>) within an hour, including the map right here on this page!

Need instructions on how to edit with iD? Here are some links to outstanding tutorials from LearnOSM, the OpenStreetMap wiki, and the United States Department of State’s Humanitarian Information Unit:

- **LearnOSM** (<http://learnosm.org/en/>)
- **OSM Beginners’ Guide** (http://wiki.openstreetmap.org/wiki/Beginners'_guide)
- **MapGive Learn to Map** (<http://mapgive.state.gov/learn-to-map/>)

Are you a mapping wiz and interested in a more advanced editor? Try out **JOSM** (<https://josm.openstreetmap.de>) with **excellent documentation** (<https://www.mapbox.com/blog/osm-mapping-guide/>) from Mapbox.

Thanks, and please check out all the posts **in the Targeted Editing series** (<https://mapzen.com/tags/targeted-editing/>)!

- **Airports** (<https://mapzen.com/blog/targeted-editing-airport-polygons>)
- **Banks** (<https://mapzen.com/blog/new-years-resolutions-money/>)
- **Building Names** (<https://mapzen.com/blog/targeted-editing-name-that-building>)
- **Campus Mapping** (<https://mapzen.com/blog/targeted-editing-campus-mapping/>)
- **Cycleways** (<https://mapzen.com/blog/targeted-editing-cycleways/>)
- **Fitness** (<https://mapzen.com/blog/new-years-resolutions-fitness>)
- **Groceries** (<https://mapzen.com/blog/new-years-resolutions-groceries>)
- **Hospitals** (<https://mapzen.com/blog/targeted-editing-hospital-polygons>)
- **Schools** (<https://mapzen.com/blog/targeted-editing-school-polygons>)
- **Stadiums & Parking** (<https://mapzen.com/blog/targeted-editing-tailgate-mania>)
- **Street Names** (<https://mapzen.com/blog/targeted-editing-no-name-roads>)
- **Transit Colours** (<https://mapzen.com/blog/targeted-editing-transit-colours>)
- **Travel & Lodging** (<https://mapzen.com/blog/new-years-resolutions-travel>)

· 04 February 2016 ·



Indy Hurt

Indy was our resident data scientist lending her geographic expertise to all things "open".

© 2017 Mapzen

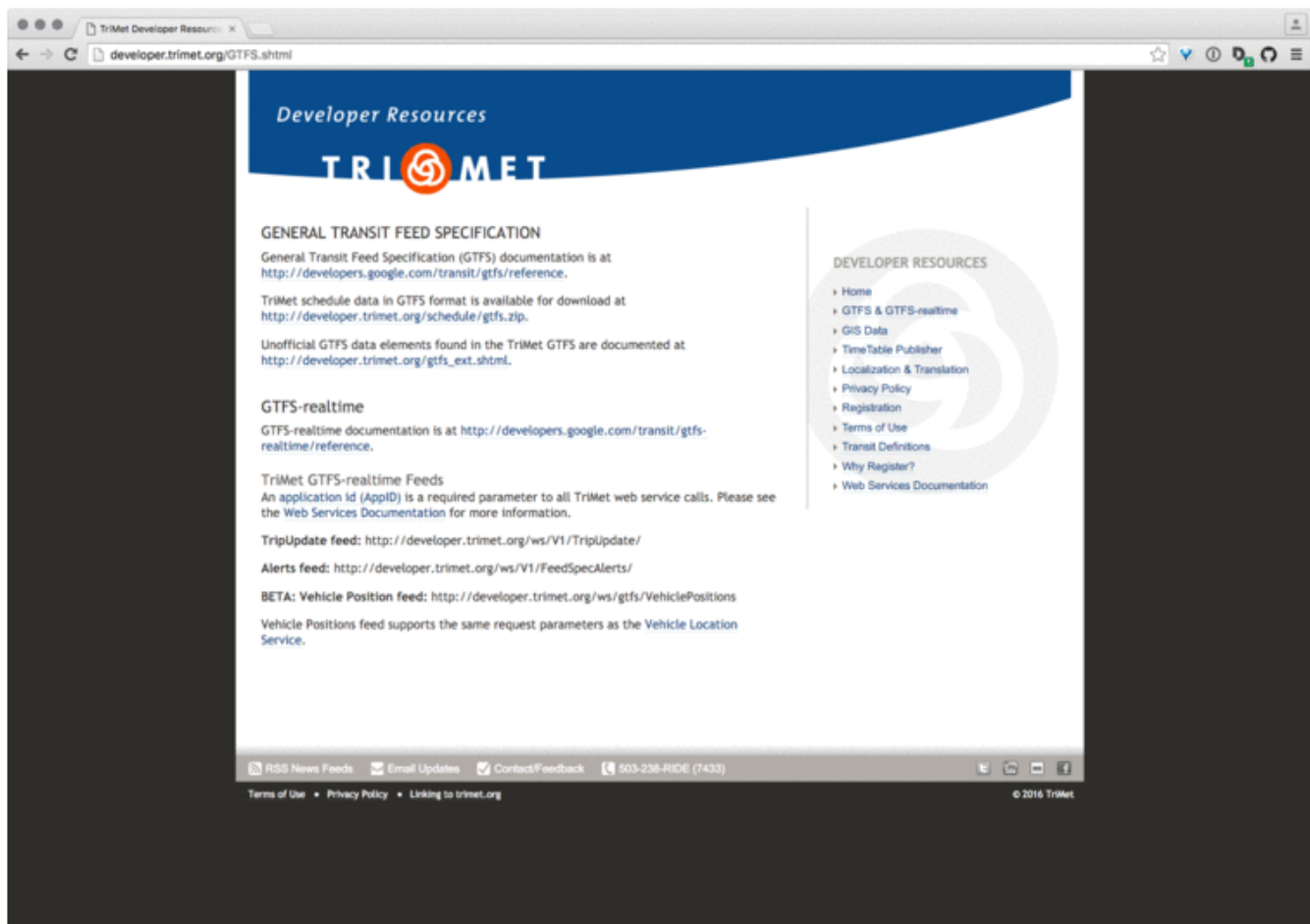
Join us as we catalog the public transit feeds of the world

transitland (/tag/transitland)

Transitland (<https://transit.land>) is an open project sponsored by Mapzen to aggregate transit data from around the world and make it seamless to use together. Back in November, we released the Transitland Feed Registry, which helps answer two questions:

- **What transit operators offer open feed? (/blog/feed-registry#q1)**
- **What can I legally do with each feed? (/blog/feed-registry#q2)**

This experiment started out in San Francisco and New York, where most Mapzen staff live, but now we're ready to open the Feed Registry to outside contributors. ***Know a feed from your part of the world or your favorite transit agency that's missing? Now you can add it to Transitland yourself!***



Transitland catalogs and imports the most popular variety of transit feeds, the **General Transit Feed Specification** (http://www.transitwiki.org/TransitWiki/index.php?title=General_Transit_Feed_Specification) format. GTFS, as it's abbreviated, is the format ingested by Google Maps, Microsoft Bing Maps, Apple Maps, and many other journey-planning services and analysis engines. Many transit operators provide GTFS feeds on their websites, and civic technologists from **Detroit** (<https://www.codeforamerica.org/projects/detroit-text-my-bus/>) to **Nairobi** (<http://www.wired.com/2015/08/nairobi-got-ad-hoc-bus-system-google-maps/>) have created and shared them as well. But these feed files are all floating around the web as individual files. The goal of Transitland is to become a center of gravity that is fully open in its software licensing, its data licensing, and its direction by a growing group of participants.

Want to help aggregate all those lonely feeds? Find a GTFS feed that's publicly available, come back to the Feed Registry and click "**add a feed** (<https://transit.land/feed-registry/feeds/new>).\" Once a feed is added and imported, it will be listed on the Transitland Feed Registry, available to visualize using the **Transitland Playground data explorer** (</blog/welcome-to-the-transitland-playground>), and open to querying using the **Datastore API** (<https://transit.land/how-it-works/#slide-3>).

Transitland is a big tent. We want to welcome software developers who are enthusiastic about data **and** public servants who do the hard work of keeping transit networks running.

Data Enthusiasts and Civic Technologists, we hope you'll find the Feed Registry "add a feed" experience addictive enough that you'll want to come back again to add a second, and a third, and a fourth, that will be useful to consume in your apps, visualizations, and analyses. **The catalog of feeds and operators that you'll be helping to build is in the public domain (<https://transit.land/an-open-project/#original-data>)**, so it will be available for you to use in whatever service you can imagine. To kick this contribution process off, Mapzen staff will be reviewing the first set of submissions that come in. As we work out the kinks, we'd appreciate involvement from others in the curation of Feed Registry submissions.

Transit Professionals, we hope you'll find the Feed Registry "add a feed" experience worth your time. It's a simple and quick way to spread your agency's existing GTFS feed out to an even wider audience—no technical changes are required if you host your GTFS feed at a stable URL on your public website. Nor does adding a feed to the Feed Registry trigger any changes to the license or legal terms that your agency has already attached to it. If you indicate how the license works, **the Feed Registry will list what users are allowed to do with your agency's feed (<https://transit.land/an-open-project/#aggregated-data>)**. We welcome any questions and are glad to support agencies with technical or legal questions. Please write us at **(<mailto:transitland@mapzen.com>)transitland@mapzen.com** (**<mailto:transitland@mapzen.com>**)

Transitland is **now open for your contributions (<https://transit.land/feed-registry/feeds/new>)**.

· 08 February 2016 ·



Drew Dara-Abrams

Drew leads Mapzen Mobility products (and aspires to being a flâneur).

© 2017 Mapzen

New Year's Resolutions – Fitness

osm (/tag/osm) **targeted-editing** (/tag/targeted-editing)

Maps are current as of Oct 2016.

Did you resolve to get in shape this year? We are nearly six weeks in, and maybe you haven't started, or maybe you have already given up! Don't worry, you are not alone. Fitness related New Year's resolutions are always very popular and *often* broken.

Business listings and other points of interest are slowly gaining momentum in OpenStreetMap. Curious to know how prevalent gyms and fitness centers are in OpenStreetmap? Check out this table featuring the same cities we presented last month in our **previous resolutions post** (<https://mapzen.com/blog/new-years-resolutions-grocery>):

	Fitness Points of Interest*	with Addresses	with Website	with Hours
Auckland	1	100%	0%	0%
Dublin	11	18%	27%	0%
Hong Kong	0	0%	0%	0%
Melbourne	45	47%	47%	13%
Mexico City	4	0%	0%	0%
Mumbai	3	0%	0%	0%
San Francisco	49	55%	22%	4%
São Paulo	5	20%	0%	0%
Seoul	1	0%	0%	0%
Singapore	4	0%	0%	0%
Stockholm	48	15%	19%	6%

	Fitness Points of Interest	with Addresses	with Website	with Hours
Vancouver	0	0%	0%	0%

*SQL queries (https://github.com/mapzen-data/targeted-editing/blob/gh-pages/queries/fitness_sql.sql) for those interested in generating stats for additional cities.

Clearly *lots* of editing opportunity!

Each one of these cities is likely to have many more fitness related businesses than what is present in OpenStreetMap. The active OpenStreetMap communities of editors may just be focused on adding other features. Of those we have found, business hours are low across the board. Not surprising when the common tag for adding hours seems complicated at first glance. We break it down in our suggestions below.

Helpful Tags

If your fitness resolution has fallen flat, get re-energized by adding or enhancing fitness related businesses in OpenStreetMap! Need some help choosing tags? Here are some suggestions:

- **name=*** (<https://wiki.openstreetmap.org/wiki/Key:name>)
- **addr:housenumber=*** (<https://wiki.openstreetmap.org/wiki/Key:addr>)
- **leisure=fitness_centre**
(https://wiki.openstreetmap.org/wiki/Tag:leisure%3Dfitness_centre)
- **website=*** (<https://wiki.openstreetmap.org/wiki/Key:website>)
- **opening_hours=*** (https://wiki.openstreetmap.org/wiki/Key:opening_hours)
 - Tips (because the opening_hours tag can be confusing)
 - Use military time.
 - YES: 06:00-22:00
 - NO: 6am to 10pm
 - Use two letter day indicators: Su Mo Tu We Th Fr Sa
 - YES: Mo-Fr
 - NO: Mon thru Fri
 - Do not add extra spaces between two letter day indicators or time ranges
 - YES: Mo-Fr 06:00-22:00
 - NO: Mo - Fr 06:00 - 22:00
 - Separate more complex hours with a semicolon
 - YES: Mo-Fr 06:00-20:00; Sa-Su 08:00-16:00
 - No: another other separator besides a semicolon!
 - Additional examples:

- 24/7
 - Mo-Fr 06:00-22:00; Sa-Su 08:00-16:00
 - Mo-Su 06:00-22:00; Th off
- Looking for a helpful interface to help your format this tag properly? Check out **YoHours** (<http://github.pavie.info/yohours/>)

Where to Start

Do you have a membership to a gym in your neighborhood? Add that one first. Do you pass by a fitness center that you are curious about on your way home from work every day? Don't be shy. Stop in and ask for information. Save those details in OpenStreetMap for future retrieval. Maybe you just need a fitness gadget or some new fitness apparel to kick your fitness resolution into overdrive! What is your go-to sporting goods store? Add it to OpenStreetMap.

Interactive Map

Would it be helpful to see where the current fitness related POIs are in OpenStreetMap? There's a map for that! Hover over any of the bright blue highlighted fitness points of interest to bring up an info bubble with links to editing tools that can be used to add additional tags. (While you pan and zoom to your neighborhood, note that fitness POIs only show up once you zoom in to level 15.)



Leaflet | Geocoding by Mapzen

This map is interactive! Open full screen ↗ (<https://mapzen-data.github.io/targeted-editing/te-fitness-centres/map/index.html?fitness>)

Not familiar with San Francisco? Search for your city! Is your go-to fitness place missing all together? Shift-click anywhere on the map to open iD or open your favorite OpenStreetMap editor to start mapping!

If you are trying to decide between creating a point or a polygon to represent your feature, a point is great when the gym or fitness centre is in a building that contains other businesses and/or residences. If your feature occupies the entire building, digitize a building polygon if it doesn't already exist and add your tags to it.

Look for more posts in the **New Year's Resolution** (<https://mapzen.com/tag/new-years-resolutions>) series soon!

Getting Started with OpenStreetMap

Need instructions on how to edit with iD? Here are some links to outstanding tutorials from LearnOSM, the OpenStreetMap wiki, and the United States Department of State's Humanitarian Information Unit:

- **LearnOSM** (<http://learnosm.org/en/>)
- **OSM Beginners' Guide** (http://wiki.openstreetmap.org/wiki/Beginners'_guide)
- **MapGive Learn to Map** (<http://mapgive.state.gov/learn-to-map/>)

Are you a mapping wiz and interested in a more advanced editor? Try out **JOSM** (<https://josm.openstreetmap.de>) with **excellent documentation** (<https://www.mapbox.com/blog/osm-mapping-guide/>) from Mapbox.

Editing with a Mobile Device

With points of interest, I bet you are interested in a mobile app to edit OpenStreetMap while you are on the go.

iOS users can check out **Go Map!!** (<https://appsto.re/us/dafwj.i>) by Bryce Cogswell. This free editor allows you to add features and edit existing features. Check out the **Go Map!!** (http://wiki.openstreetmap.org/wiki/Go_Map!!) wiki page for more details. **Pushpin** (<http://www.pushpinosm.org>) by Fulcrum is another excellent iOS editor for OpenStreetMap points of interest.

Looking for an Android equivalent? **Vespucci** (<https://play.google.com/store/apps/details?id=de.blau.android>) by Marcus Wolschon is a great option. Check out the **Vespucci** (<http://vespucci.io>) home page for documentation and tutorials.

Careful, now. Adding and enhancing features in OpenStreetMap is fun, but to burn lots of calories editing, you'll need a treadmill desk. Get outside, increase your local knowledge, and share it with the world through your OpenStreetMap edits!

*Check out all the posts in the **Targeted Editing series** (<https://mapzen.com/tag/targeted-editing>):*

- **Airports** (<https://mapzen.com/blog/targeted-editing-airport-polygons>)
- **Banks** (<https://mapzen.com/blog/new-years-resolutions-money/>)
- **Building Names** (<https://mapzen.com/blog/targeted-editing-name-that-building>)
- **Campus Mapping** (<https://mapzen.com/blog/targeted-editing-campus-mapping/>)
- **Cycleways** (<https://mapzen.com/blog/targeted-editing-cycleways/>)
- **Fitness** (<https://mapzen.com/blog/new-years-resolutions-fitness>)
- **Groceries** (<https://mapzen.com/blog/new-years-resolutions-groceries>)

- **Hospitals** (<https://mapzen.com/blog/targeted-editing-hospital-polygons>)
- **Schools** (<https://mapzen.com/blog/targeted-editing-school-polygons>)
- **Stadiums & Parking** (<https://mapzen.com/blog/targeted-editing-tailgate-mania>)
- **Street Names** (<https://mapzen.com/blog/targeted-editing-no-name-roads>)
- **Transit Colours** (<https://mapzen.com/blog/targeted-editing-transit-colours>)
- **Travel & Lodging** (<https://mapzen.com/blog/new-years-resolutions-travel>)

· 11 February 2016 ·



Indy Hurt

Indy was our resident data scientist lending her geographic expertise to all things "open".

© 2017 Mapzen

I Am Here

whosonfirst (/tag/whosonfirst)

I Am Here

tl;dr

<https://whosonfirst.mapzen.com/iamhere/> (<https://whosonfirst.mapzen.com/iamhere/>) is a shiny new version of **Simon Willison's** (<http://blog.simonwillison.net/>) classic "Get Lat Lon" application **full of Mapzen-y goodness** (<https://github.com/whosonfirst/whosonfirst-www-iamhere>).

A short history

In late 2007 **Simon Willison** (<http://blog.simonwillison.net/>) launched what some people have described as "the most useful website on the internet". The website was called **Get Lat Lon** (<https://web.archive.org/web/20071013001113/http://getlatlon.com/>) and its entire purpose was to enable a visitor to "find the latitude and longitude of a point on a map. "

The website was built using **the Google Maps API** (<https://developers.google.com/maps/>) and had a form for geocoding addresses or place names but the primary interface was a simple map with a set of crosshairs centered in the viewport. Get Lat Lon would simply print the geographic coordinates of whatever location happened to be beneath the crosshairs. Brilliant!

Somewhere between 2007 and now the domain renewal for Get Lat Lon lapsed and now it's... something else entirely, something not worth linking to. You can still get a feeling for the simplicity and elegance of its overall design because there are snapshots of the website in the **Wayback Machine** (<https://web.archive.org/>) except... none of the Javascript works anymore.

INTERNET ARCHIVE

WayBackMachine

111 captures

13 Oct 07 - 15 Jan 13

Go

Get Lat Lon

Find the latitude and longitude of a point on a map.

Place name:

Zoom to place

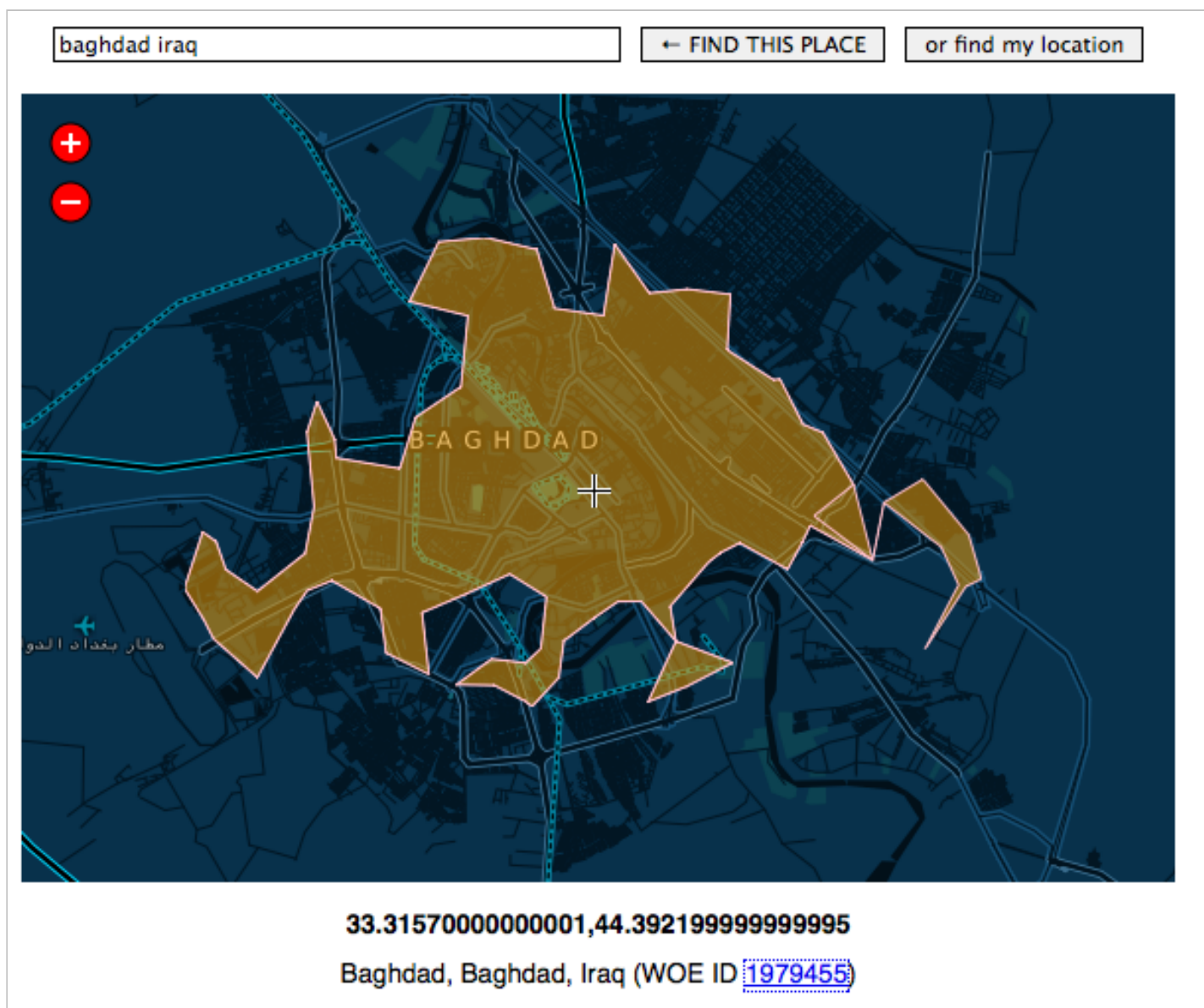
Latitude, Longitude:

Google Maps zoom level:

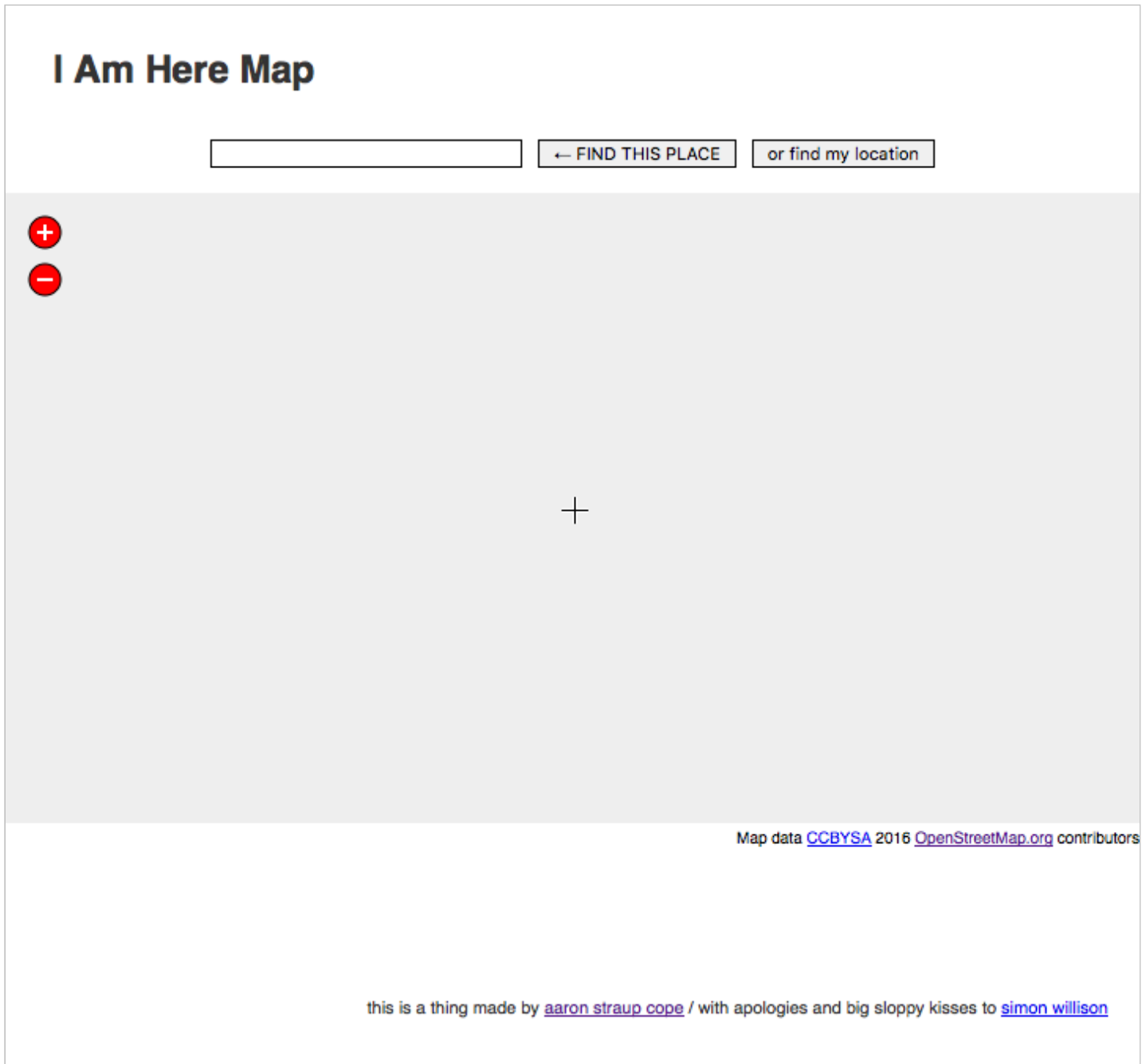
Built by [Simon Willison](#)

In 2009 I decided to write my own version of Get Lat Lon. Instead of using the Google Maps API it would use all "open" software and data. The map data would be from **OpenStreetMap** (<http://www.openstreetmap.org/>). The map tiles would be **from CloudMade using exciting new cartography from Stamen Design** (<http://mike.teczno.com/notes/cloudmade-styles.html>). It would use the **modestmaps.js** (<https://github.com/stamen/modestmaps-js>) library for managing all those tiles. It would support the then still-nascent browser-based **Geolocation API** (<http://www.w3.org/TR/geolocation-API/>) to help determine your location. The geocoding would **be handled by Flickr**

(<https://www.flickr.com/services/api/flickr.places.find.html>) and in addition to geocoding it would also try to *reverse* geocode your location and display **the shape of the place** (<http://code.flickr.net/2008/10/30/the-shape-of-alpha/>) contained by a latlon, again **using the Flickr API** (<https://www.flickr.com/services/api/flickr.places.findByLatLon.html>).



And... it would be **so clever and modular** (<https://github.com/straup/js-iamheremap>) that it would support multiple service providers and you could just drop it in to any webpage and it would work, as if like magic. It was called "I Am Here" and I think I was the only person to ever use it but **it's still running** (<http://www.aaronland.info/iamhere/>). In 2014, though, CloudMade **got out of the tile business** (<https://wiki.openstreetmap.org/wiki/CloudMade>) and so there is literally not much to see anymore.



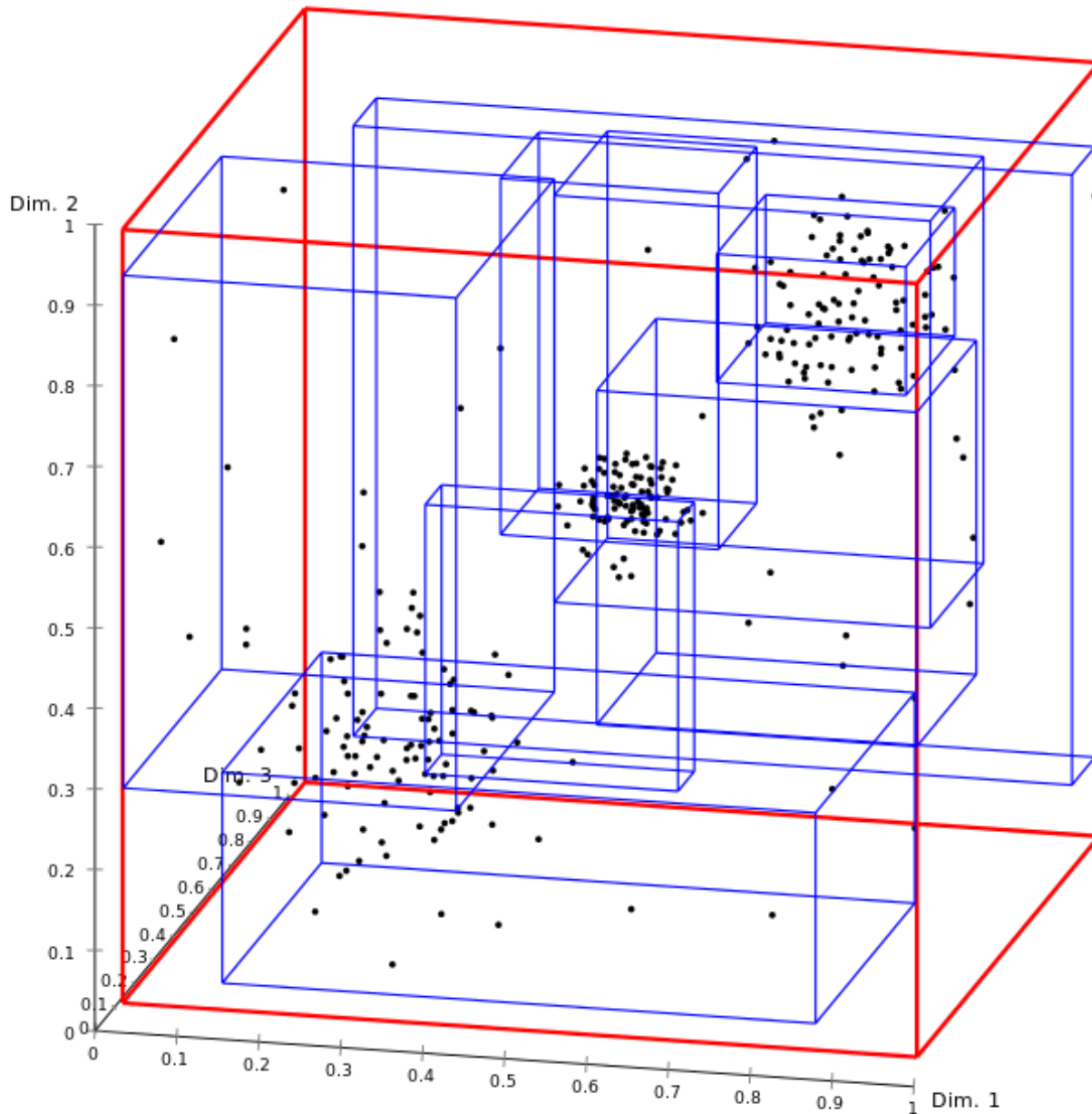
I am pretty sure that it's exactly **one line of code to define a new map provider** (<https://github.com/straup/js-iamheremap/blob/master/iamheremap.src.js#L268-L280>) to make I Am Here work again but to be perfectly honest just looking at all that too-too clever code now, in 2016, is exhausting. Also, see the way the licensing information on the map data hasn't been updated to reflect **the switch to the ODbL** (<http://www.openstreetmap.org/copyright>)...

Reverse geocoding

Fast forward to last year (2015) and work has begun in earnest on the **Who's On First (WOF) gazetteer** (<https://whosonfirst.mapzen.com/>) at Mapzen. Part of that work has been to build hierarchies for each record in the gazetteer which is something of a **chicken-and-egg problem** (<https://github.com/whosonfirst/whosonfirst-placetypes#here-is-a-pretty-picture>). We've been automating the process with a general purpose **point-in-polygon tool** (<https://github.com/whosonfirst/go-whosonfirst-pip/>) that we've written in-house using the Go programming language. It is called `go-whosonfirst-pip` and it works like this:

- It loads an arbitrary number of WOF documents in to memory. *Where "WOF document" just means any GeoJSON document with a properties dictionary containing `id` , `name` and `placetype` keys.*
- The documents are indexed using an **R-tree** (<https://en.wikipedia.org/wiki/R-tree>). *An R-tree is a data structure optimized for storing multi-dimensional information, like geometries.*
- The document set is queried with a latitude and a longitude and optionally filtered by placetype.
- Because the R-tree stores bounding boxes instead of complex geometries (as Wikipedia says: *The "R" in R-tree is for rectangle*) there is a final operation (called **raycasting** (https://en.wikipedia.org/wiki/Ray_casting)) to ensure that a point is actually contained by any of the candidate results.

That's it. The purpose of the `go-whosonfirst-pip` code is to do fiddly math across a large and heterogenous dataset as quickly as possible.



(<https://commons.wikimedia.org/w/index.php?curid=10008216>)

*This is what an R-tree looks like, courtesy **Wikimedia user Chire***

(<https://commons.wikimedia.org/wiki/User:Chire>)

It only knows about points and things that contains those points but it does not know about context. For example, consider the following question: What continent is **Russia** (<https://whosonfirst.mapzen.com/spelunker/id/85632685/>) a part of? Europe? Asia? All of the above? There are lots of interesting applications that remain to be built on top of `go-whosonfirst-pip` but it is important to remember that it is *not* an inference engine, by design.

The code includes a simple HTTP server (called `wof-pip-server`) that you can use to easily load (and then query) one or more **“meta” files**

(<https://github.com/whosonfirst/whosonfirst-data/tree/master/meta>) containing pointers to different WOF documents. If a WOF document is just a GeoJSON with a few explicit properties then a “meta” file is just a CSV with a `path` column containing a *relative* path to a WOF document.

Although the “meta” files were originally conceived as little more than a simple helper tool (or index) for large volumes of data they have grown in to something of a “first class” object inside the world of Who’s On First, with more and more of the tooling and infrastructure built around them. They are due for a longer more detailed discussion but not today.

To get started with an instance of `wof-pip-server` that will query for countries and neighbourhoods you would do:

```
$> ./bin/wof-pip-server -data /usr/local/mapzen/whosonfirst-data/data/ \
    /usr/local/mapzen/whosonfirst-data/meta/wof-country-latest.csv \
    /usr/local/mapzen/whosonfirst-data/meta/wof-neighbourhood-latest.csv
[placetype] country 219
[placetype] neighbourhood 49906
```

Depending on how fast your computer is the indexing process might take a couple of minutes. By default the `wof-pip-server` listens for requests on port `8080` on your computer’s local “loopback” network interface which is also called `localhost` , so the URL for querying the server would be `http://localhost:8080` . For example:

```
$> curl 'http://localhost:8080?latitude=40.677524&longitude=-73.987343' | python -mjson.tool
[
  {
    "Id": 102061079,
    "Name": "Gowanus Heights",
    "Placetype": "neighbourhood"
  },
  {
    "Id": 85633793,
    "Name": "United States",
    "Placetype": "country"
  },
  {
    "Id": 85865587,
    "Name": "Gowanus",
    "Placetype": "neighbourhood"
  }
]
```

If you want to limit the result set to a specific placetype simply append `placetype=PLACETYPE` to your query string, like this:

```
$> curl 'http://localhost:8080?latitude=40.677524&longitude=-73.987343&placetype=neighbourhood' | python -mjson.tool
[
  {
    "Id": 102061079,
    "Name": "Gowanus Heights",
    "Placetype": "neighbourhood"
  },
  {
    "Id": 85865587,
    "Name": "Gowanus",
    "Placetype": "neighbourhood"
  }
]
```

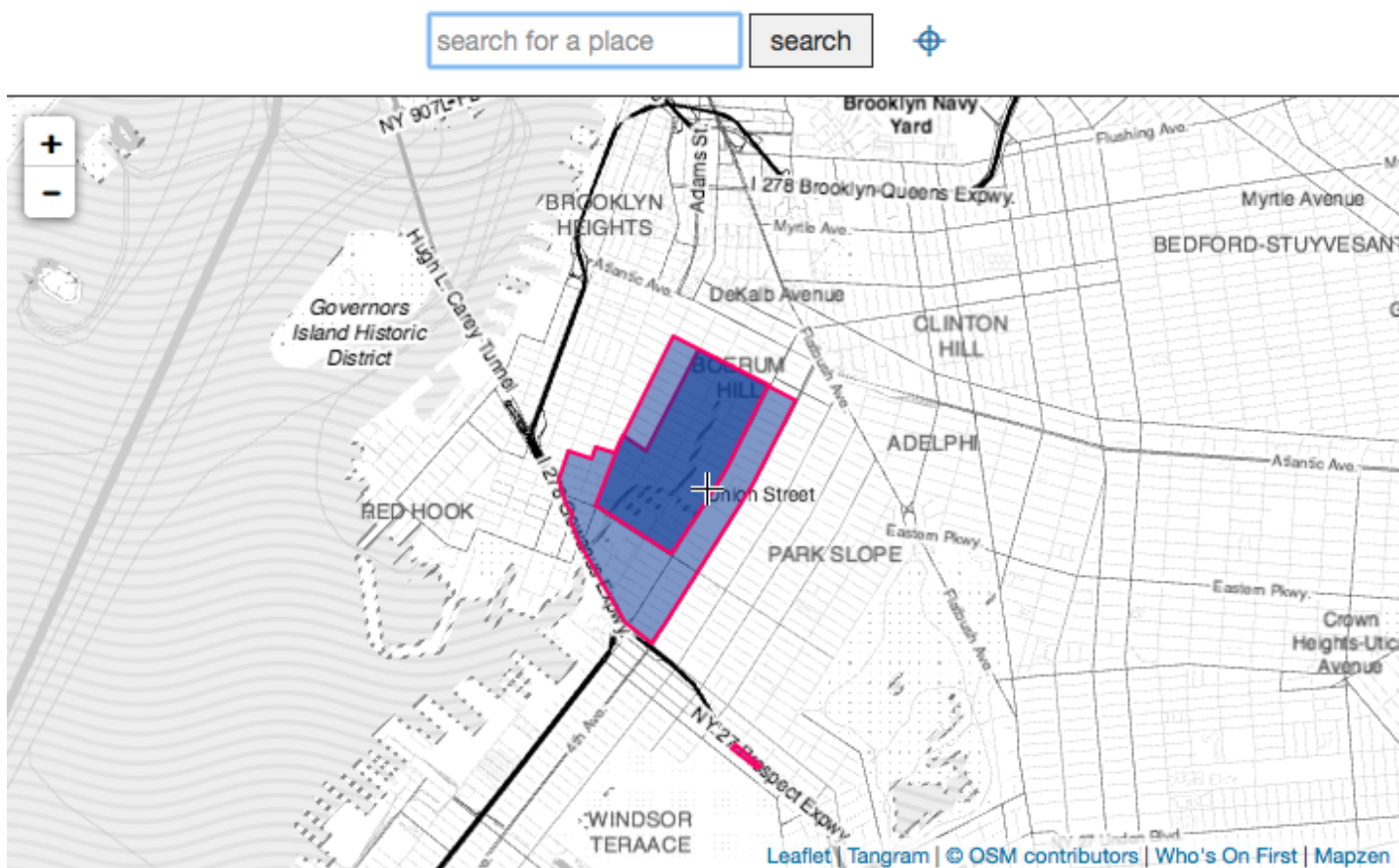
Currently it is **not possible** (<https://github.com/whosonfirst/go-whosonfirst-pip/issues/22>) to filter the result set with multiple placetypes. That's not technically a bug but it's become clear that it's also not a feature.

The `wof-pip-server` returns as little information as possible because it stores as little information as possible, mostly for performance reasons. It is left up to applications using `wof-pip-server` to decide whether and how to look up more information about any given WOF document.

The nice thing about the `go-whosonfirst-pip` tools is that they are designed to be agnostic as possible about the data they index and serve. For example I recently downloaded **version 2.0.1 of the Flickr Alpha Shape files** (<https://archive.org/details/FlickrShapesPublicDataset2.0.1.tar>) and re-jiggled the file structure (but not the actual data) of the alpha shapes and now they will Just Work™ with `wof-pip-server` .

whosonfirst-www-iamhere

So far, so good. We have **an enormous bag of (Who's On First) data** (<https://github.com/whosonfirst/whosonfirst-data/>) and we have **a tool for establishing the relationship(s) between those files** (<https://github.com/whosonfirst/go-whosonfirst-pip/>) but any volume of geographic data absent a map is... hard to see.



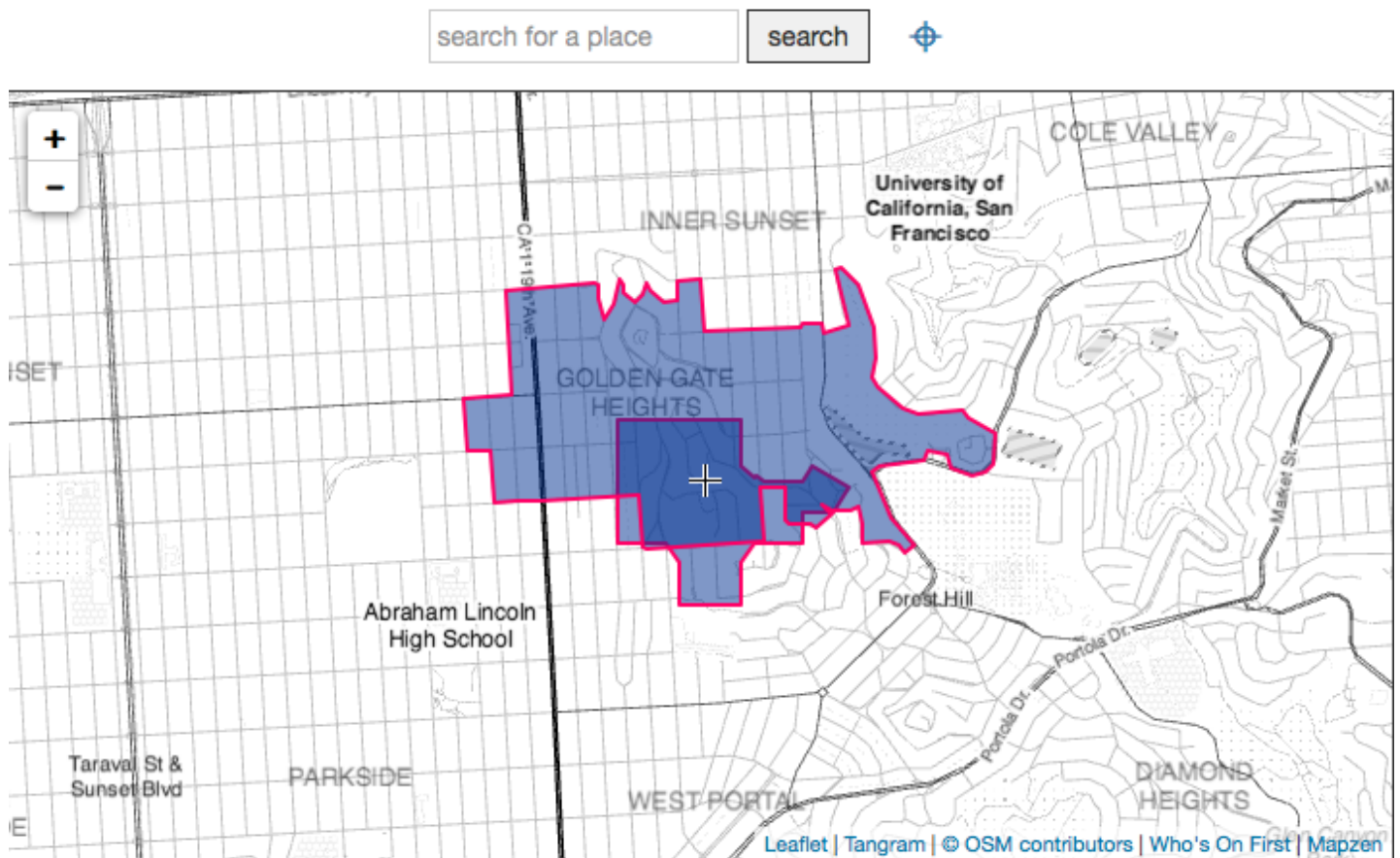
the map is centered at **40.676277,-73.988371** at zoom level **13** which appears to be somewhere in **Gowanus** or **Gowanus Heights**

(<https://whosonfirst.mapzen.com/iamhere/#14/40.676277/-73.988371>)

So, I rebuilt I Am Here (or Get Lat Lon) ... again. It's called **whosonfirst-www-iamhere** (<https://github.com/whosonfirst/whosonfirst-www-iamhere>) or just I Am Here (again) for short.

It does everything that the combination of Get Lat Lon and the original I Am Here did, but is built entirely using Mapzen tools and services.

Aside from an ongoing need to simply know what the coordinates are for any given spot on a map it seemed like `whosonfirst-www-iamhere` would be a good and useful tool for visualizing and sanity-checking the results returned by the `go-whosonfirst-pip` code.



the map is centered at **37.751376,-122.469149** at zoom level **14** which appears to be somewhere in **Golden Gate Heights or Golden Gateheights**

(<https://whosonfirst.mapzen.com/iamhere/#14/37.751376/-122.469149>)

For example, Golden Gate...what?

This time, though, it's been built with two guiding principles in mind. The first is that "Mapzen should always be Consumer Zero (of Mapzen services)" and the second is to minimize the pain and nuisance of any one piece, of what is actually a pretty complex application, failing or shutting down or otherwise going offline.

Mapzen as Consumer Zero

The latest version of I Am Here uses a bunch of Mapzen services already:

- It uses the **Tangram** (<https://github.com/tangrams/tangram>) Javascript library for rendering tiles and **Refill** (<https://github.com/tangrams/refill-style>) for styling them.
- It uses **Mapzen Search** (<https://mapzen.com/projects/search>), for geocoding. *This is not enabled by default as you'll need to **sign up for an API key** (<https://mapzen.com/developers>)*

(it's really easy!) and add it to the `mapzen.whosonfirst.config.js` config file.

- It uses **Who's On First data** (<https://whosonfirst.mapzen.com/data/>) for reverse geocoding and for displaying geometries and other metadata about a place.

In time, it will also use:

- **Mapzen Turn-by-Turn** (<https://mapzen.com/projects/valhalla>) to visualize the placetypes along a journey route. *There is **a branch of the `go-whosonfirst-pip` code** (<https://github.com/whosonfirst/go-whosonfirst-pip/tree/polyline>) that will perform point-in-polygon tests along an **encoded polyline** (<https://developers.google.com/maps/documentation/utilities/polylinealgorithm>) as returned by the Turn-by-Turn service so this feature is mostly waiting on user-interface details rather than number-crunching.*
- A **Who's On First enabled IP lookup service** (<https://whosonfirst.mapzen.com/mmdb/>) to help determine where to position the map when I Am Here is launched.

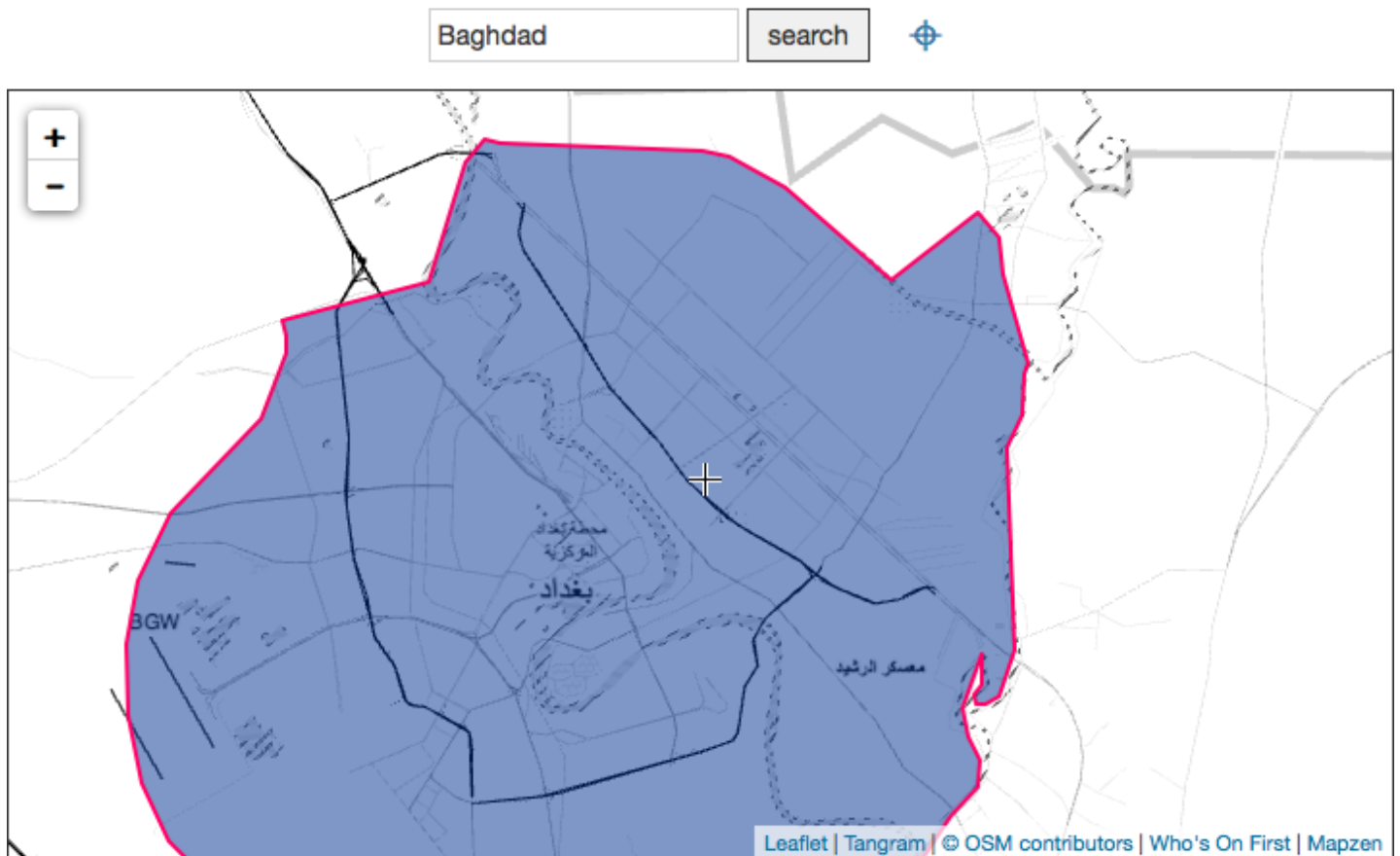
Small pieces, loosely failing

The ultimate goal of `whosonfirst-www-iamhere` is to work from your own computer in offline-mode (or when you don't have a network connection) without needing to download and install a long list of dependencies. As of this writing:

- It *does not* come with pre-installed WOF data, but that's also a deliberate choice. We'll talk more about that in a moment.
- It *does not* come with pre-installed or cached vector tiles (used by Tangram). *There is **a margins-of-the-day project to enable tile-caching in Tangram** (<https://github.com/thisisaaronland/tangram/tree/localforage>) but it is not ready for general usage yet.*
- It *does not* come with a pre-installed version of Mapzen Search for offline use. *Since the code that powers the service (Pelias) is open source and designed to be run by an individual that piece is left as an exercise to the user.*
- It *does* come with pre-installed platform specific (OS X, Linux and Windows) binary applications for serving the Tangram Javascript code and the `wof-pip-server` endpoint.
- It *does* come with a handy "startup" tool meant to take care all of the details when launching `whosonfirst-www-iamhere`. *Currently the tool is written in Python which comes pre-installed on OS X and Linux computers. For people using Windows computers installing Python fails the yet-another-dependency test so there is added impetus for rewriting it (the startup tool) as something can be pre-compiled to run on a Windows machine, too.*
- It *does not* come with a handy GUI startup tool. *Currently it assumes a level of familiarity with your computer's command-line (or terminal) interface.*

Here is an example of how you might start `whosonfirst-www-iamhere` from your computer. This assumes that you have downloaded (or cloned) the **whosonfirst-www-iamhere** (<https://github.com/whosonfirst/whosonfirst-www-iamhere>) code and have navigated in to the root directory.

```
$> ./bin/start.py -d /path/to/your/whosonfirst-data/data \
/path/to/your/whosonfirst-data/meta/wof-neighbourhood-latest.csv \
/path/to/your/whosonfirst-data/meta/wof-locality-latest.csv
```



the map is centered at **33.332823,44.426651** at zoom level **11** which appears to be somewhere in **Baghdad**

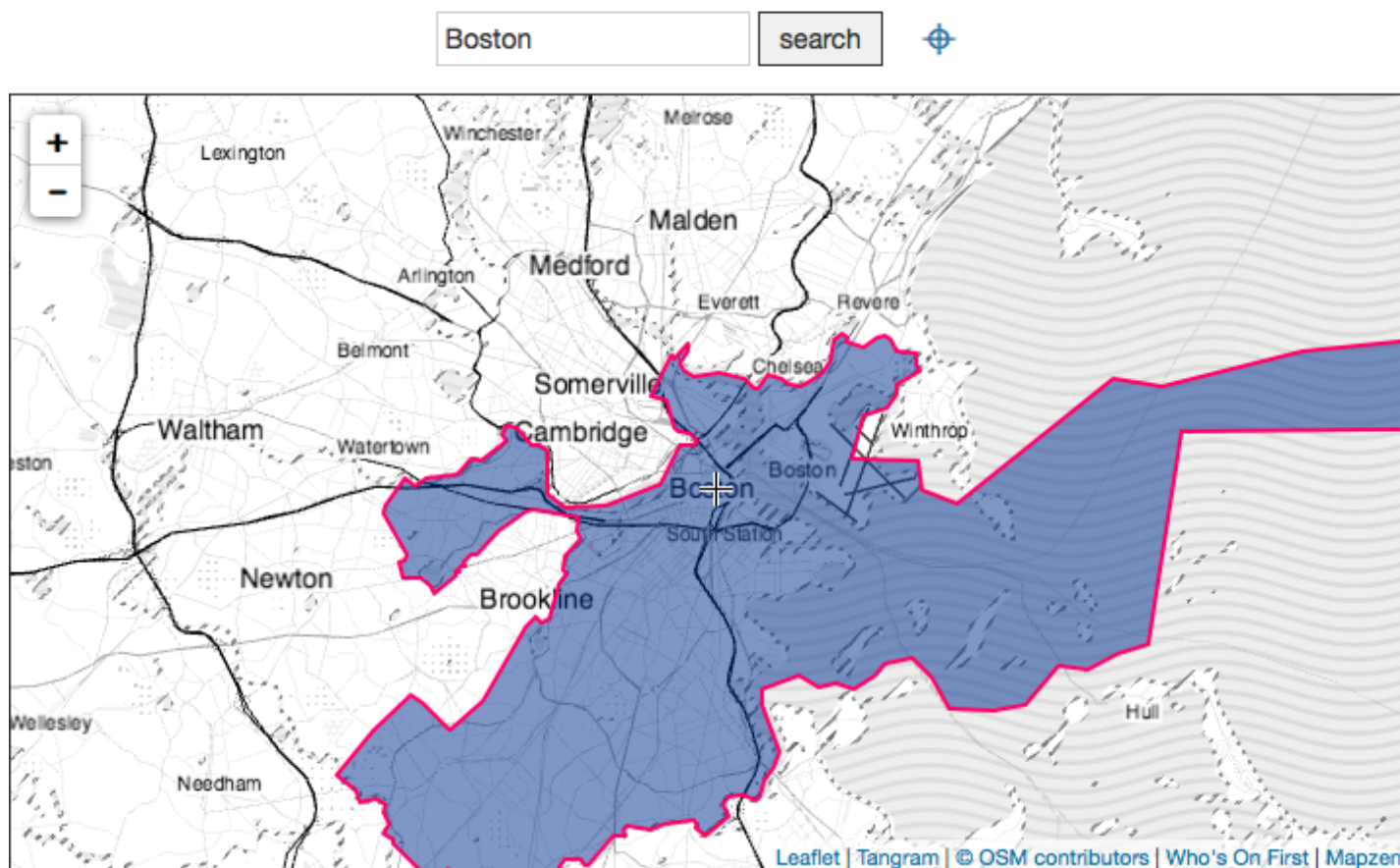
(<https://whosonfirst.mapzen.com/iamhere/#11/33.332823/44.426651>)

The short version is that once the `start.py` script has finished setting everything up you can open your web browser up at `http://localhost:8001` and start poking around countries and neighbourhoods from Who's On First.

The longer version follows. By default the `start.py` tool requires a minimum of two arguments. The first (`-d`) is the path to where, on your computer, you've stored your raw Who's On First data files. The second and third arguments are the paths to "meta" files (remember them?) that the `wof-pip-server` will index. The `start.py` tool will start three separate servers running on your computer:

- A simple web server that will the Who's On First data you specified (with the `-d` parameter) running on port `9999` . *Remember the way we said that `wof-pip-server` returns little more than a WOF ID? The reason for this data-only server is that the `whosonfirst-www-iamhere` application will "inflate" a WOF ID, returned by the PIP server, by fetching its record from the data server.*
- Another simple web server that will host the `whosonfirst-www-iamhere` application running on port `8001` . *These two pieces could be served by a single more sophisticated web server.*
- Finally the `wof-pip-server` itself running on port `8080` .

All of these port numbers can be changed if necessary. To do so you would pass your own setting as parameters to the `start.py` tools and as custom settings in the `mapzen.whosonfirst.config.js` config file.

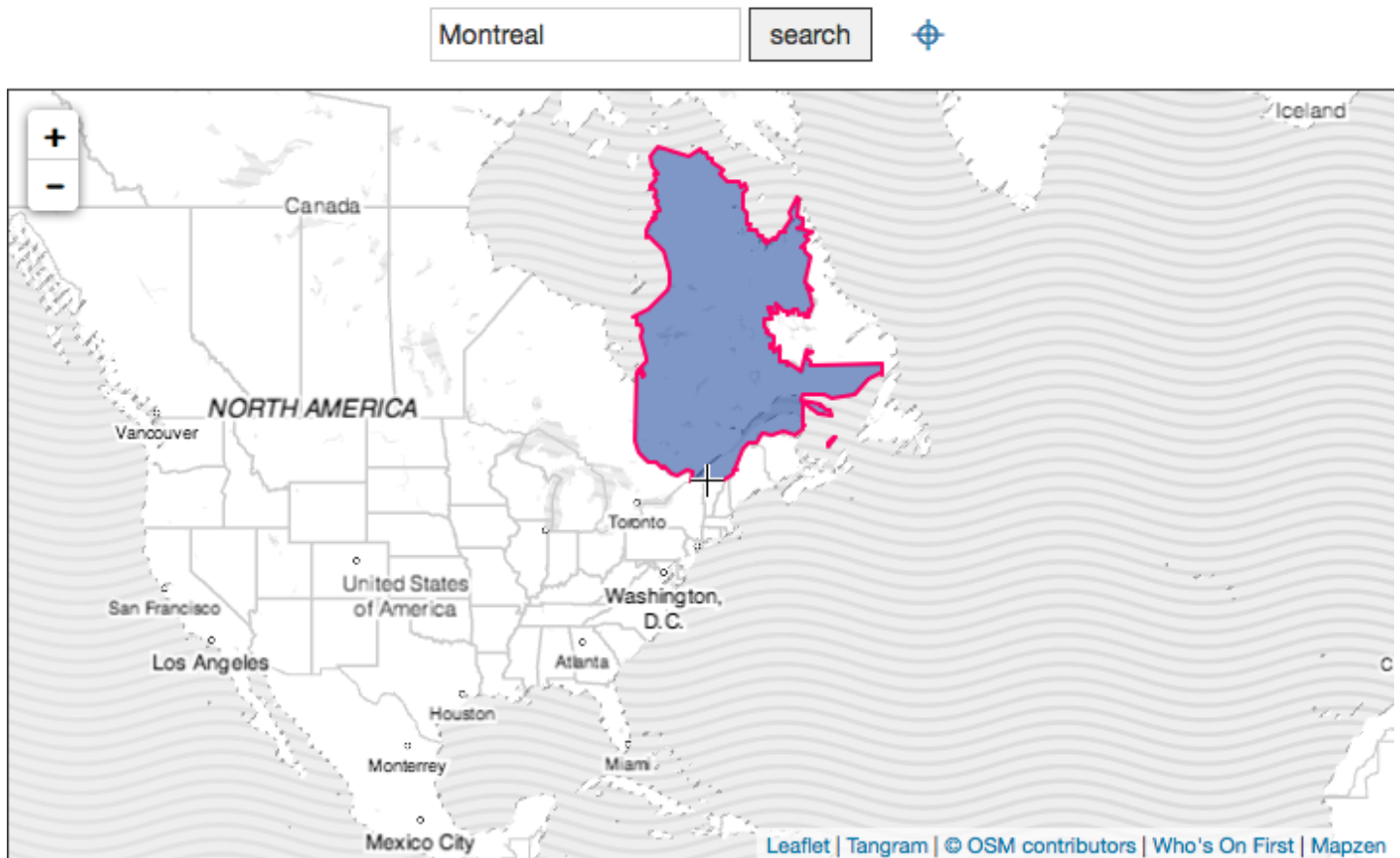


the map is centered at **42.358430,-71.059770** at zoom level **11** which appears to be somewhere in **Boston**

(<https://whosonfirst.mapzen.com/iamhere/#11/42.358430/-71.059770>)

Bundles

One of the challenges with Who's On First has been balancing our desire for a robust and portable data format (plain text GeoJSON files), the needs for an historical audit trail and the mechanics of working with and distributing a large and ever-growing dataset. We have been using Git and GitHub extensively for much of the work to date but as the commit history around **the data ()** grows so too does the size of the `whosonfirst-data` repository and the burden in using it or simply getting started with Who's On First.



the map is centered at **45.398450,-73.476562** at zoom level **3** which appears to be somewhere in **Quebec**

(<https://whosonfirst.mapzen.com/iamhere/#3/45.5080/-73.6166>)

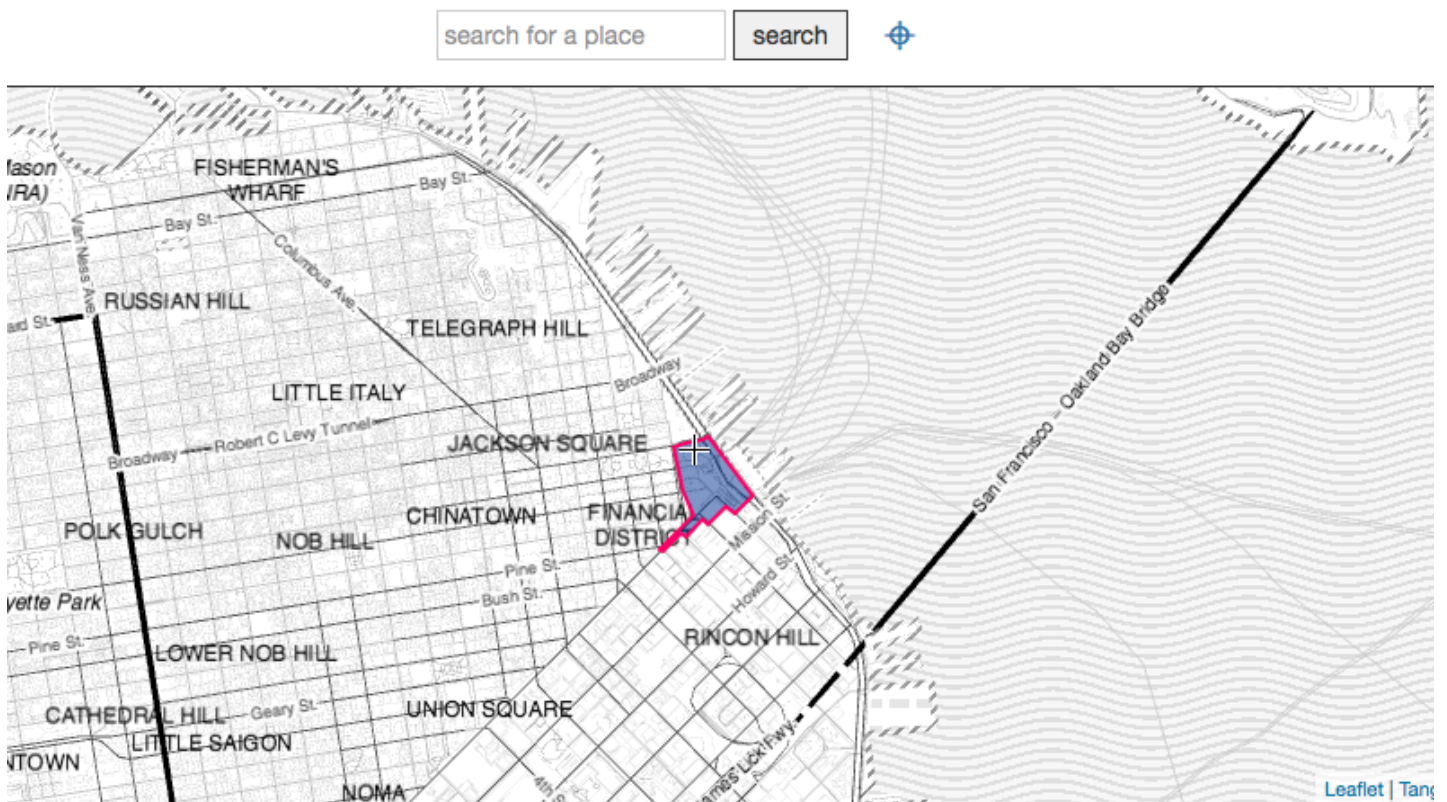
As an alternative we have been working on something called "bundles". Bundles are:

...a collection of GeoJSON formatted files (Who's On First data) grouped by a specific property, like placetype. They allow for people to more easily bulk download a subset of the entire Who's On First dataset. Currently there are only bundles by placetype but eventually we will add a variety of different "slices" of the data as demand and interest require.

Each bundle contains a "meta" file (*see... they just keep popping up all over the place!*) and a folder named `data` which contains the files listed in the meta file. Bundles do not contain any Git history or related metadata but our hunch is that many people don't need or want that information. The startup tool mentioned above does not yet have support for bundles but that will happen shortly. In the meantime you can get started with `whosonfirst-www-iamhere` and bundles with a few short commands in your terminal.

For example, if you just wanted to run a copy of `whosonfirst-www-iamhere` using only **microhoods** (<https://whosonfirst.mapzen.com/spelunker/placetypes/microhood/>) (which are currently **all in San Francisco** (<https://whosonfirst.mapzen.com/spelunker/id/85922583/descendants/?&placetype=microhood>)) you would do the following:

```
$> cd /path/to/your/whosonfirst-www-iamhere
$> curl -O https://whosonfirst.mapzen.com/bundles/wof-microhood-latest-bundle.tar.bz2
$> tar -xvjf wof-microhood-latest-bundle.tar.bz2
$> ./bin/start.py -d wof-microhood-latest-bundle/data wof-microhood-latest-bundle/wof-mic
```



the map is centered at **37.796356,-122.396107** at zoom level **14** which appears to be somewhere in **Super Bowl City**

(<https://whosonfirst.mapzen.com/iamhere/#14/37.796356/-122.396107>)

All of the details and currently available bundles are listed over at <https://whosonfirst.mapzen.com/bundles> (<https://whosonfirst.mapzen.com/bundles>) and... yes, **Super Bowl City** (<https://whosonfirst.mapzen.com/spelunker/id/420561633/>) is a thing that really happened in 2016.

Or you can just use our version

As mentioned at the beginning of this blog post there is a publicly accessible version of I Am Here for you to play with at **<https://whosonfirst.mapzen.com/iamhere/>** (**<https://whosonfirst.mapzen.com/iamhere/>**).

Right now it only display neighbourhoods but shortly **we will add the ability to select different (even multiple) placetypes to display at the same time** (**<https://github.com/whosonfirst/go-whosonfirst-pip/issues/22>**). And as circumstance permits we will add the additional features (routing and IP lookups) mentioned above. And then all the stuff we haven't even thought of yet.

Enjoy!

· 19 February 2016 ·



Aaron Straup Cope

Aaron is happiest in the presence of olive oil.

© 2017 Mapzen

opzworks for Opsworks

engineering (/tag/engineering)

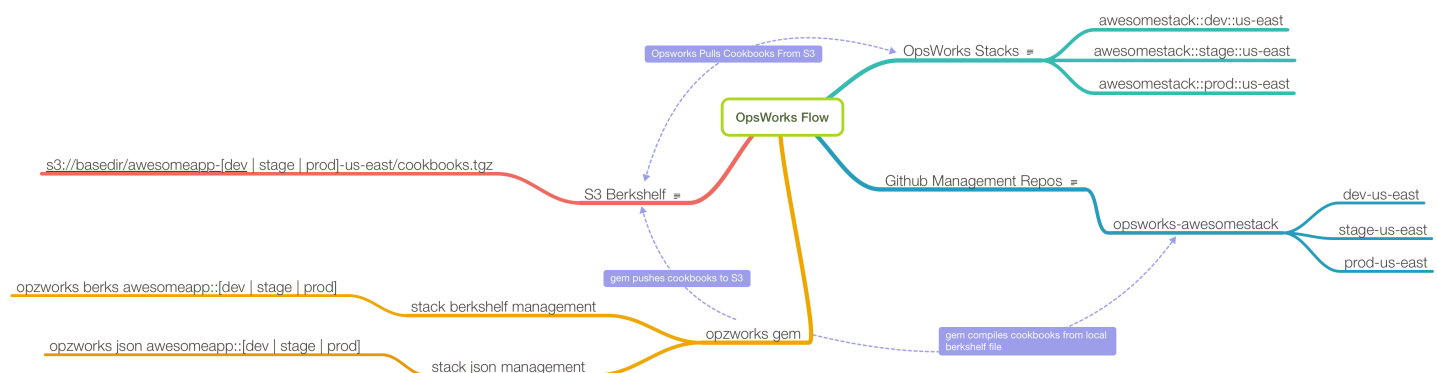
Impetus

The third post in our **Engineering at Mapzen** series (<https://mapzen.com/tag/engineering>) describes how we use **AWS Opsworks** (<https://aws.amazon.com/opsworks/>) to manage the majority of our EC2 resources and the software we run on them. It provides us with a very flexible environment where we can accomodate the range of applications we deal with, without having to run and manage Chef server.

On the flip side of that, we do need to deal with cookbooks for each individual stack we run, which is why we wrote **opzworks** (<https://github.com/mapzen/opzworks>). Specifically, opzworks allows you to:

- manage a berkshelf for each of your Opsworks stacks as part of a git repository
- manage the json for each of your stacks as part of a git repository
- generate SSH configs from some or all of your stacks

The **opzworks Wiki** (<https://github.com/mapzen/opzworks/wiki>) is quite detailed, so please check it out! There's an **A-Z Example** (<https://github.com/mapzen/opzworks/wiki/Example>) of all the setup with a new stack, as well as writeups on configuration and on using the SSH functionality.



(<https://mapzen-assets.s3.amazonaws.com/images/opzworks-for-opsworks/flow.png>)

Why Opsworks

One great upside of AWS Opsworks is automatic asset tagging. Every EC2 resource launched in an Opsworks stack is tagged with an **opzworks:stack** key. As you start to grow, this means using AWS cost analysis tools to get a breakdown of where you're spending money becomes trivial. Simply run a report grouped by stack and you can see which projects are eating into that budget more than perhaps you'd like.

Another thing we like about Opsworks is resource association. For example, RDS instances, ECS clusters and ELBs can all be associated with a stack, making it easy to see at a glance what a particular stack setup is comprised of. Also, don't discount the Opsworks web interface as unimportant... while all of Opsworks is underpinned with AWS API calls, being able to navigate systems visually to see running instances, associated resources, attached load balancers and monitoring information is all hugely beneficial.

In a company with varying degrees of operational experience, having a platform that allows enormous amounts of flexibility is of great importance. We can manage all our resources in a central location, but still allow people to:

- deploy code by clicking on a button
- write code to automate deployments via API
- deploy a simple rails app with no custom cookbook code using Opsworks built in layers and cookbooks
- write custom cookbook code that manages a large data workflow
- manage access to entire stacks via IAM policies
- manage users (ssh keys)

What need does opzworks fill?

When you start using custom cookbooks with AWS Opsworks, you immediately realize that if you want individual stacks to reference their own store of cookbooks, rather than pull from a common shared repository, you're going to need a utility to help manage that process. This was the original need that opzworks was written to fill: to simply build a tarball of cookbooks derived from a Berksfile, upload it to S3, and tell the stack instances to refresh their local cache of cookbooks.

The original implementation was config file driven, which meant whenever you added a new stack, you'd need to create a config file that referenced its stackId. The current implementation, however, derives this information for itself, which means you can simply pass it a stack name and it will infer or retrieve via API all the information required to do its job.

Convention & Workflow with opzworks

The use of opzworks requires adherence to a **workflow** (<https://github.com/mapzen/opzworks/wiki/Workflow>) with somewhat rigid naming conventions: for stacks, that means **name::env::region**. Whether you're creating Opsworks stacks by hand or via CloudFormation, the requirement to adhere to a standard stack naming convention and cookbook repository structure is a good thing, especially seeing as how a lot of the chef code we write takes advantage of programmatic access to the stack name.

The next convention follows from the first, which is that for each project, there will be one **opzworks-project** github repo with branches for each **environment-region**. So if, for example, we have an elasticsearch project with dev and prod envs in both us-east and us-west, you'd want to see...

Opsworks stacks with the following names:

- elasticsearch::prod::us-east
- elasticsearch::prod::us-west
- elasticsearch::dev::us-east
- elasticsearch::dev::us-east

and corresponding branches in an **opzworks-elasticsearch** github repo for this project:

- prod-us-east
- prod-us-west
- dev-us-east
- dev-us-west

In each branch of the repo you'll maintain a **Berksfile** and a **stack.json** that corresponds to the Opsworks stack.

Now, whenever we want to build a new cookbook bundle for, say, dev::us-east, we can run:

- `opzworks berks elasticsearch::dev::us-east`

This will use berksshelf to create a cookbook bundle and upload it to S3, and at the end of the run trigger `update_custom_cookbooks` on the stack so that all running instances have the latest cookbook code.

Similarly, stack json would be updated with:

- `opzworks json elasticsearch::dev::us-east`

You'll be shown a diff of your existing stack json compared with the json you've updated in your repository, and asked to confirm the push.

SSH

If you use Opsworks currently and don't want to re-arrange things in support of this workflow, you still might be interested in `opzworks` for the SSH config functionality:

`opzworks ssh` will generate a config file based off all the instances in your stack suitable for use in `~/.ssh/config`. Alternatively, it can operate on individual stacks:

```
opzworks ssh elasticsearch::dev::us-east
```

Or on all stacks matching a pattern:

```
opzworks ssh elasticsearch
```

There are a few more options, namely the ability to return either private or public IPs, or to just return IPs rather than a formatted config file.

Please check out the **wiki** (<https://github.com/mapzen/opzworks/wiki>) for all the details! And if you liked this post, **check out the rest of our engineering series** (<https://mapzen.com/tag/engineering>).

· 23 February 2016 ·



Grant Heffernan

Grant is a sysadmin with fingers in all of Mapzen's pies. Egli non si senta italiano, ma per fortuna o purtroppo...

Inside Libpostal – a fast, multilingual, international street address parser trained on OpenStreetMap data

*For the past year, data scientist **Al Barrentine** (<https://twitter.com/albarrentine>) has been working with Mapzen to crack one of the hardest problems in geocoding and place search: international address parsing. It's resulted in **Libpostal** (<https://github.com/openvenues/libpostal>), a state-of-the-art, lightning-fast C library and statistical model for parsing and normalizing addresses around the world. The address parser alone is 98.9% accurate. And by virtue of being written in C, libpostal can be used directly from several popular languages, with bindings already written for **Python** (<https://github.com/openvenues/pypostal>), **Go** (<https://github.com/openvenues/gopostal>), **Ruby** (https://github.com/openvenues/ruby_postal), **Java** (<https://github.com/openvenues/jpostal>), and **NodeJS** (<https://github.com/openvenues/node-postal>).*

*The world is a big place, but Libpostal is a big step toward making it easier to find any place anywhere (and it only uses open data). We at Mapzen are incredibly excited to soon be using Libpostal as a key part of **Mapzen Search** (<https://mapzen.com/projects/search>) and we can't wait to see what you use it for!*

Here, Al explains just how Libpostal came to be and, importantly, shares how it works so others can benefit from what he learned.

Street addresses are among the more **quirky** (<https://www.mjt.me.uk/posts/falsehoods-programmers-believe-about-addresses/>) artifacts of human language, yet they are crucial to the increasing number of applications involving maps and location. Last year I worked on a collaboration with **Mapzen** (<https://mapzen.com/>) with the goal of building smarter, more international geocoders using the vast amounts of local knowledge in open geographic data sets.

The result is **libpostal** (<https://github.com/openvenues/libpostal>): a multilingual street address parsing/normalization library, written in C, that can handle addresses all over the world.

Libpostal uses machine learning and is informed by tens of millions of real-world addresses from **OpenStreetMap** (<http://www.openstreetmap.org/>). The entire pipeline for training the models is open source. Since OSM is a dynamic data set with thousands of contributors and the models are retrained periodically, improving them can be as easy as contributing addresses to OSM.

Each country's addressing system has its own set of conventions and peculiarities and libpostal is designed to deal with practically all of them. It currently supports normalizations in 60 languages and can parse addresses in more than 100 countries. Geocoding using libpostal as a preprocessing step becomes drastically simpler and more consistent internationally.

The core library is written in pure C, which means that in addition to having a small carbon footprint, libpostal can be used from almost any stack or programming language. There are currently bindings written for **Python** (<https://github.com/openvenues/pypostal>), **Go** (<https://github.com/openvenues/gopostal>), **Ruby** (https://github.com/openvenues/ruby_postal), **Java** (<https://github.com/openvenues/jpostal>), and **NodeJS** (<https://github.com/openvenues/node-postal>) with more popular languages coming soon.

But let's rewind for a moment.

Why we care about addresses

Addresses are the **unique identifiers** (<https://mapzen.com/blog/meaningful-geocoding-address-search-the-two-core-principles-of-geocoding>) humans use to describe places, and are at the heart of virtually every facet of modern Internet-connected life: map search, routing/directions, shipping, on-demand transportation, delivery services, travel and accommodations, event ticketing, venue ratings/reviews, etc. There's a \$1B company in almost every one of those categories.

The central information retrieval problem when working with addresses is known as geocoding. We want to transform the natural language addresses that people use to describe places into lat/lon coordinates that all our awesome mapping and routing software uses.

Geocoding's not your average document search. Addresses are typically very short strings, highly ambiguous, and chock full of abbreviations and local context. There is usually only one correct answer to a query from the user's perspective (with the exception of broader searches like

“restaurants in Fort Greene, Brooklyn”). In some instances we may not even have the luxury of user input at all e.g. batch geocoding a bunch of addresses obtained from a CSV file, the Web or a third-party API.

Despite these idiosyncrasies, we tend to use the same full-text search engines for addresses as we do for querying traditional text documents. Out of the box, said search engines are terrible at indexing addresses. It's easy to see how a naïve implementation could pull up addresses on St Marks Ave when the query was “St Marks Pl” (both the words “Ave” and “Pl” have a low **inverse document frequency** (<https://en.wikipedia.org/wiki/Tf%E2%80%93idf>) and do not affect the rank much). Autocomplete might yield addresses on the 300 block of Main Street for a query of “30 Main Street”. Abbreviations like “Saint” and “St” which are not simple prefix overlaps might not match in most spellcheckers since their **edit distance** (https://en.wikipedia.org/wiki/Levenshtein_distance) is greater than 2.

Typically we employ all sorts of heuristics to help with address matching: synonyms lists, special tokenizers, analyzers, regexes, simple parsers, etc. Most of these methods require changing the search engine's config, and make US/English-centric, overly-simplified assumptions. Even using a full-text search engine in general won't help in the server-side batch geocoding case unless we're fully confident that the first result is the correct one.

Geocoding in 2016

Libpostal began with the idea that geocoding is more similar to the problem of **record linkage** (https://en.wikipedia.org/wiki/Record_linkage) than text search.

The question we want to be able to answer is: “are two addresses referring to the same place?” Having done that, we can simultaneously make automated decisions in the batch setting and return more relevant results in user-facing geocoders.

This decomposes into two sub-problems:

1. **Normalization:** the easiest way to handle all the abbreviated variations and ambiguities in addresses is to produce canonical strings suitable for machine comparison, i.e. make “30 W 26th St” equal to “Thirty West Twenty-Sixth Street”, and do it in every language.
2. **Parsing:** some components of an address are more essential than others, like house numbers, venue names, street names, and postal codes. Beyond that, addresses are highly structured and there are multiple redundant ways of specifying/qualifying them. “London, England” and “London, United Kingdom” specify the same location if parsed to mean

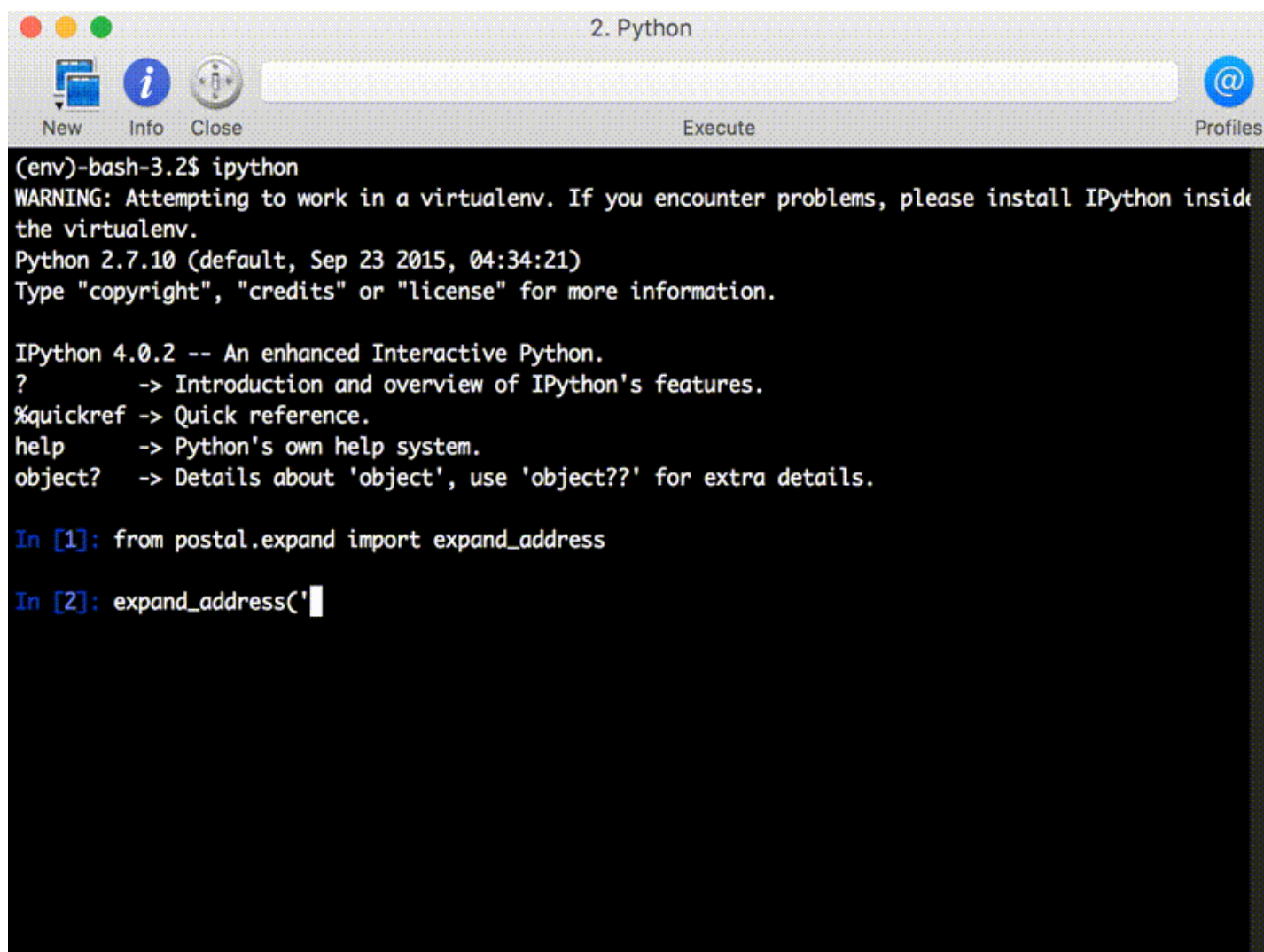
city/admin1 and city/country respectively. If we already know London, there would be no point in returning addresses in Manchester simply because it's also in the UK.

Once we've got canonical address strings segmented into components, geocoding becomes a much simpler string matching problem, the kind that full-text search engines and even relational/non-relational databases are good at handling. With a little finesse one could conceivably geocode with nothing but libpostal and a hash table.

To see how that's possible, the next two sections describe in detail how libpostal addresses (pun very much intended) the normalization and parsing problems respectively.

Multilingual address normalization

Normalization is the process of converting free-form address strings encountered in the wild into clean normalized forms suitable for machine comparison. This is primarily deterministic/rule-based.



```
(env)-bash-3.2$ ipython
WARNING: Attempting to work in a virtualenv. If you encounter problems, please install IPython inside
the virtualenv.
Python 2.7.10 (default, Sep 23 2015, 04:34:21)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.2 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]: from postal.expand import expand_address

In [2]: expand_address('
```

There are several steps involved in making normalization work across so many different languages. I'll mention the notable ones.

Multilingual tokenization

Tokenization is the process of segmenting text into words and symbols. It is the first step in most NLP applications, and there are many **nuances**

(<https://www.ibm.com/developerworks/community/blogs/nlp/entry/tokenization?lang=en>). The tokenizer in libpostal is actually a **lexer**

(https://en.wikipedia.org/wiki/Lexical_analysis) implementing the Unicode Consortium's **TR-29 spec** (<http://unicode.org/reports/tr29/>) for unicode word segmentation. This method handles every script/alphabet, including ideograms (used in languages not separated by whitespace e.g. Chinese, Japanese, Korean), which are read one character at a time.

The tokenizer is inspired by the approach in Stanford's **CoreNLP**

(<https://github.com/stanfordnlp/CoreNLP>) i.e. write down a bunch of regular expressions and use compile them into a fast **DFA**

(https://en.wikipedia.org/wiki/Deterministic_finite_automaton). We use **re2c**

(<http://re2c.org/>), a light-weight scanner generator which often produces C that's as fast as a handwritten equivalent. Indeed, tokenization is quite fast, chunking through > 2 million tokens per second.

Abbreviation expansion

Almost every language on Earth uses abbreviations in addresses. Historically this had to do with width constraints on things like street signs or postal envelopes. Digital addresses face similar constraints, namely that they are more likely than other types of text to be viewed on a mobile device.

Abbreviations create ambiguity, as there are multiple ways of writing the same address with different degrees of verbosity: "W St Johns St", "W Saint Johns St", "W St Johns Street", and "West Saint Johns Street" are all equivalent. There are similar patterns in most languages.

For expanding abbreviations to their canonical forms, libpostal contains a number of **per-language dictionaries**

(<https://github.com/openvenues/libpostal/tree/master/resources/dictionaries>), which are simple text files mapping "Rd" to "Road" in 60 languages. Each word/abbreviation can have one or more canonical forms ("St" can expand to "Street" or "Saint" in English), and one or more dictionary types: directionals, street suffixes, honorifics, venue types, etc.

Dictionary types make it possible to control which expansions are used, say if the input address is already separated into discrete fields, or if using libpostal's address parser to the same effect. With dictionary types, it's possible to apply only the relevant expansions to each component. For instance, in an English address, "St." always means "Saint" when used in a city or country name like "St. Louis" or "St. Lucia" and will only be ambiguous when used as part of a street or venue/building name.

The dictionaries are compiled into a **trie** (<https://en.wikipedia.org/wiki/Trie>) data structure, at which point a fast search algorithm is used to scan through the string and pull out matching phrases, even if they span multiple words (e.g. "State Route"). This type of search also allows us to treat multi-word phrases as single tokens during address parsing.

Ideographic languages like Japanese and Korean are handled correctly, even though the extracted phrases are not surrounded by whitespace. So are Germanic languages where street suffixes are often appended onto the end of the street name, but may optionally be separated out (Rosenstraße and Rosen Straße are equivalent). All of the abbreviations listed on the **OSM Name Finder wiki** (http://wiki.openstreetmap.org/wiki/Name_finder:Abbreviations) are implemented as of this writing, plus many more.

At the moment, libpostal does not attempt to resolve ambiguities in addresses, and often produces multiple potential expansions. Some may be nonsensical ("Main St" expands to both "Main Street" and "Main Saint"), but the correct form will be among them. The outputs of libpostal's `expand_address` can be treated as a set and address matching can be seen as a doing a set intersection, or a JOIN in SQL parlance. In the search setting, one should index all of the strings produced, and use the same code to normalize user queries before sending them to the search server/database.

Future iterations of `expand_address` will probably use OpenStreetMap (where abbreviation is discouraged) to build classifiers for ambiguous expansions, and include an option for outputs to be ranked by likelihood. This should help folks who need a "single best" expansion e.g. when displaying the results on a map.

Address language classification

Abbreviations are language-specific. Consider expanding the token "St." in an address of unknown language. The canonical form would be "Sankt" in German, "Saint" in French, "Santo" in Portuguese, and so on.

We don't actually want to list all of these permutations. In most user-facing geocoders, we likely know the language ahead of time (say from the user's HTTP headers or current location). However, in batch geocoding, we don't know the language of any of our input addresses, so will need a classifier to predict languages automatically using only the input text.

Language detection is a well-studied problem and there are several existing implementations (such as **Chromium's compact language detector** (<https://github.com/CLD2Owners/cld2>)) which achieve very good results on longer text documents such as Wikipedia articles or webpages. Unfortunately, because of some of the aforementioned differences between addresses and other forms of text, packages like CLD which are trained on webpages usually expect more/longer words than we have in an abbreviated address, and will often get the language wrong or fail to produce a result at all.

So we'll need to build our own language classifier and train it specifically for address data. This is a **supervised learning** (https://en.wikipedia.org/wiki/Supervised_learning) problem, which means we'll need a bunch of address-related input labeled by language, like this:

```
de Graf-Folke-Bernadotte-Straße
sv Tollare Träskväg
nl Johannes Vermeerstraat Akersloot
it Strada Provinciale Ca' La Cisterna
da Østervang Vissenbjerg
nb Lyngtangen Egersund
en Wood Point Road
ru улица Солунина
ar جادة صائب سلام
fr Rue De Longpré
he השרון
ms Jalan Sri Perkasa
cs Jeřabinová Rokycany
ja 山口秋穂線
ca Avinguda Catalunya
es calle Camilo Flammarion
eu Mungialde etorbidea
pt Rua Pedro Muler Faria
```

Sounds great, but where are we going to find such a data set? In libpostal, the answer to that question is almost always: use OpenStreetMap.

OSM has a great system when it comes to languages. By default the name of a place is the official local language name, rather than the Anglicized/Latinized name. **Beijing** (<http://www.openstreetmap.org/node/25248662>)'s default name for instance is “北京市” rather than “Beijing” or “Peking.”

Some addresses in OSM are explicitly labeled by language, especially in countries with multiple official street sign languages like Hong Kong, Belgium, Algeria, Israel, etc. In cases where a single name is used, we build an **R-tree** (<https://en.wikipedia.org/wiki/R-tree>) polygon index that can answer the question: for a given lat/lon, which official and/or regional language(s) should I expect to see? In Germany we expect addresses to be in German. In some regions of Spain, Catalan or Basque or Galician will be returned as the primary language we expect to see on street signs, whereas (Castilian) Spanish is used as a secondary alternative. In cases where languages are equally likely to appear, the language dictionaries in libpostal are used to help disambiguate. Lastly, street signs are always be written in the languages spoken by the majority of people, a vestige of **linguistic imperialism** (https://en.wikipedia.org/wiki/Linguistic_imperialism), and the language index accounts for this as well.

All said and done, this process produces around 80 million language-labeled address strings. From there we extract **features** ([https://en.wikipedia.org/wiki/Feature_\(machine_learning\)](https://en.wikipedia.org/wiki/Feature_(machine_learning))) (informative attributes of the input which help to predict the output) similar to those used in Chromium and the language detection literature: sequences of 4 letters or 1 ideogram, whole tokens for words shorter than 4 characters, and a shortcut for **unicode scripts** ([https://en.wikipedia.org/wiki/Script_\(Unicode\)](https://en.wikipedia.org/wiki/Script_(Unicode))) mapping to a single language like Greek or Hebrew. Specific to our use case, we also include entire phrases matching certain language dictionaries from libpostal.

We then train a **multinomial logistic regression** (http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression) model (also known as softmax regression) using **stochastic gradient descent** (<http://leon.bottou.org/publications/pdf/compstat-2010.pdf>) and a few sparsity **tricks** (<http://research.microsoft.com/pubs/192769/tricks-2012.pdf>) to keep training times reasonably fast. Logistic regression is heavily used in NLP because unlike Naïve Bayes, it does not make the assumption that input features are independent, which is unrealistic in language.

Another nice property of logistic regression is that its output is a well-calibrated probability distribution over the labels, not just normalized scores that look like probabilities if you “close one eye and squint with the other.” With real probabilities we can implement meaningful decision boundaries. For instance, if the top language returned by the classifier has a probability

of 0.99, we can safely ignore the other language dictionaries, whereas if it makes a less confident prediction like 0.62 French and 0.33 Dutch, we might want to throw in both dictionaries. Though the latter type of output should *not* be interpreted as the distribution of languages in the address itself (as in a multi-label classifier), results with multiple high-probability languages are most often returned in cases like Brussels where addresses actually are written in two languages.

Numeric expression parsing

In many addresses, particularly on the Upper East Side of Manhattan it seems, numbers are written out as words e.g. “Eighty-sixth Street” instead of “86th Street.” Libpostal uses a simplified form of the **Rule-based Number Format (RBNF)**

(<http://www.unicode.org/repos/cldr/trunk/common/rbnf/>) in CLDR which spells out the grammatical rules for parsing/spelling numbers in various languages.

Rather than try to exhaustively list all numbers and ordinals that might be used in an address, we supply a handful of rules which the system can then use to parse arbitrary numbers.

In English, when we see the word “hundred”, we multiply by any number smaller than 100 to the left and add any number smaller than 100 to the right. There’s a recursive structure there. If we know the rule for the hundreds place, and we know how to parse all numbers smaller than 100, then we can “count” up to 1000.

Numeric spellings can get reasonably complicated in other languages. French for instance uses some Celtic-style numbers which switch to base 20, so “quatre-vingt-douze” (“four twenties twelve”) = 92. Italian numbers rarely contain spaces so “milleottocentodue” = 1802. In Russian, ordinal numbers can have 3 genders. Libpostal parses them all, currently supporting numeric expressions in over 30 languages.

Roman numerals can be optionally recognized in any language (so “IX” normalizes to 9), though they’re most commonly found in Europe in the titles of popes, monarchs, etc. In most cases Roman numerals are the canonical form, and can be ambiguous with other tokens (a single “I” or “V” could also be a person’s middle initial), so a version of the string with unnormalized Roman numerals is added as well.

Transliteration

Many addresses around the world are written in a non-Latin scripts such as Greek, Hebrew, Cyrillic, Han, etc. In these cases, addresses can be written in the local alphabet or **transliterated** (<https://en.wikipedia.org/wiki/Transliteration>) i.e. converted to a Latin script equivalent.

Because the target script is usually Latin, transliteration is also sometimes known as “Romanization.”

For example, “Тверская улица” in Moscow transliterates to “Tverskaya ulitsa.” A restaurant website would probably use the former for its Russian site and the latter for its international site. Street signs in many countries (especially those who’ve at some point hosted a World Cup) will typically list both versions, at least in major cities.

Libpostal takes advantage of all the transliterators available in the Unicode Consortium’s **Common Locale Data Repository (CLDR)** (<http://cldr.unicode.org/translation/transforms>), again compiling them to a trie for fast runtime performance. The implementation is lighter weight than having to pull in **ICU** (<http://site.icu-project.org/>), which is a huge dependency and may conflict with system versions.

Each script or script/language combination can use one or more different transliterators. There are for instance several differing standards for transliterating Greek or Hebrew, and libpostal will try them all.

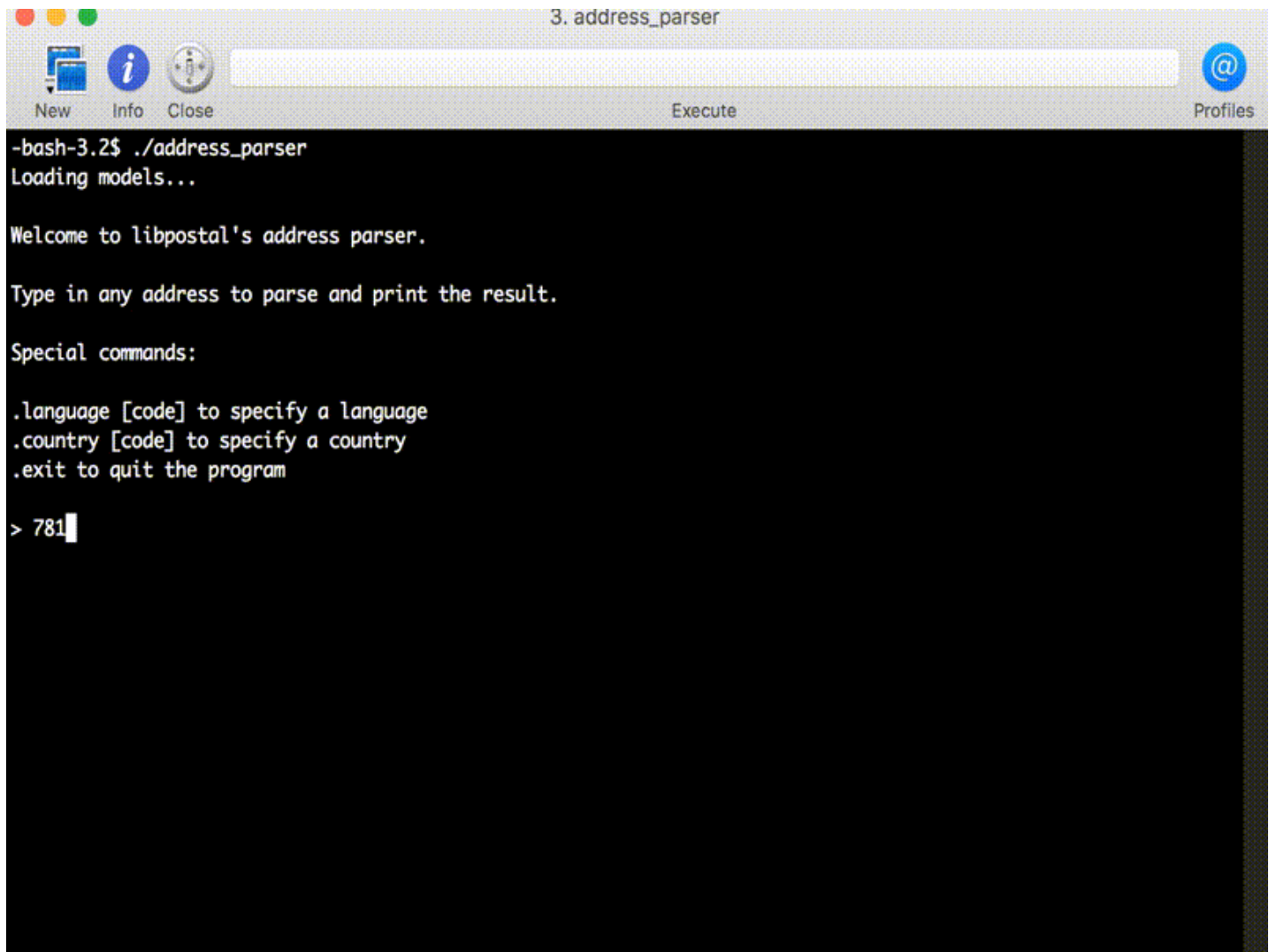
There’s also a simpler transliterator, the **Latin to ASCII** (<http://unicode.org/repos/cldr/trunk/common/transforms/Latin-ASCII.xml>) transform, which converts “œ” to “oe”, etc. This is in addition to standard **Unicode normalization** (https://en.wikipedia.org/wiki/Unicode_equivalence), which would decompose “ç” into “c” and “COMBINING CEDILLA (U+0327)”, and optionally strip the diacritical mark to make it just “c.” Accent stripping is sort of an “ignorant American” type of normalization, and can change the pronunciation or meaning of words. Still, sometimes addresses have to be written in an ASCII approximation (because keyboards), especially with travel-related searching, so we do strip accent marks by default, with an optional flag to prevent it.

Some countries actually translate addresses into English (something like “Tverskaya Street”), creating further ambiguity. At the cost of potentially adding a few bogus normalizations, libpostal can handle such translations by simply adding English dictionaries as a second language option for certain countries/languages/scripts.

International address parsing

Parsing is the process of segmenting an address into components like house number, street name, city, etc. Though many address parsers have been written over the years, most are rule-based and only designed to handle US addresses. In libpostal we develop the first NLP-based

address parser that works well internationally.



```
-bash-3.2$ ./address_parser
Loading models...

Welcome to libpostal's address parser.

Type in any address to parse and print the result.

Special commands:

.language [code] to specify a language
.country [code] to specify a country
.exit to quit the program

> 781
```

The NLP approach to address parsing

International address parsing is something we could never possibly hope to solve deterministically with something like regex. It might work reasonably well for one country, as addresses tend to be highly structured, but there are simply too many variations and ambiguities to make it work across languages. This sort of problem is where machine learning, particularly in the form of **structured learning** (https://en.wikipedia.org/wiki/Structured_prediction), really shines.

Most NLP courses/tutorials/libraries focus on models and algorithms, but applications on real-world data sets are not in great abundance. Libpostal provides an example of what an end-to-end production-quality NLP application looks like. I'll detail the relevant steps of the pipeline below, all of which are open source and published to Github as part of the repository.

Creating labeled data from OSM

OpenStreetMap addresses are already separated into components. Here's an example of OSM tags as JSON:

```
{
  "addr:housenumber": "30",
  "addr:postcode": "11217",
  "addr:street": "Lafayette Avenue",
  "name": "Brooklyn Academy of Music"
}
```

This is exactly the kind of output we want our parser to produce. These addresses are hand-labeled by humans and there are lots of them, more than 50 million at last count.

We want to construct a supervised tagger, meaning we have labeled text at training time, but only unlabeled text (geocoder input) at runtime. The input to a sequence model is a list of tagged tokens. Here's an example of the for the address above:

```
Brooklyn/HOUSE Academy/HOUSE of/HOUSE Music/HOUSE
30/HOUSE_NUMBER Lafayette/ROAD Avenue/ROAD Brooklyn/CITY NY/STATE 11217/POSTCODE
```

At runtime, we'll only expect to see "Brooklyn Academy of Music, 30 Lafayette Avenue, Brooklyn, NY 11217", potentially without the commas. With a little creativity, we can reconstruct the free-text input, and tag each token to produce the above training example.

Notice that the original OSM address has no structure/ordering, so we'll need to encode that somewhere. For this, we can use OpenCage's **address-formatting** (<https://github.com/OpenCageData/address-formatting>) repo, which defines address templates for almost every country in the world, with coverage increasing steadily over time. In the US, house number comes before street name ("123 Main Street"), whereas in Germany or Spain it's the inverse ("Calle Ruiz, 3"). The address templates are designed to format OSM tags into human-readable addresses in every country. This is a good approximation of how we expect geocoder input to look in those countries, which means we have our input strings. I've personally contributed a few dozen countries to the repo and it's getting better coverage all the time.

Also notice that in the OSM address, city, state, and country are missing. We can "fill in the blanks" by checking whether the lat/lon of the address is contained in certain administrative polygons. So that we don't have to look at every polygon on Earth for every lat/lon, we construct

an **R-tree** (<https://en.wikipedia.org/wiki/R-tree>) to quickly check bounding box containment, and then do the slower, more thorough point-in-polygon test on the bounding box matches. The polygons we use are a mix of OSM relations, Quattroshapes/GeoNames localities, and Zetashapes for neighborhoods.

Making the parser robust

Because geocoders receive a wide variety of queries, we then perturb the address in several ways so the model has to train on many different kinds of input. With certain random probabilities, we use:

- **Alternate names:** for some of the admin polygons (e.g. “NYC”, “New York”, “New York City”) so the model sees as many forms as possible
- **Alternate language names:** OSM does a great job of handling language in addresses. By default a tag like “name” can be assumed to be in the local official language, or hyphenated if there’s more than one language. Something like “name:en” would be the English version. In countries with multiple official languages like Hong Kong, addresses almost always have per-language tags. We use these whenever possible.
- **Non-standard polygons:** like boroughs, counties, districts, neighborhoods, etc. which may be occasionally seen in addresses
- **ISO codes and state abbreviations:** so the parser can recognize things like “Berlin, DE” and “Baltimore, MD”
- **Component dropout:** we usually produce 2–3 different versions of the address with various components removed at random. This way the model also has to learn to parse simple “city, state” queries alongside venue addresses, so it won’t get overconfident e.g. that the first token in an address is always a venue name.

Structured learning

In the structured learning, we typically use a linear model to predict the most likely tag for a particular word given some local and contextual attributes or features. What differentiates structured learning from other types of machine learning is that in structured learning, the model’s prediction for the previous word can be used to predict the current word. In similar tasks like part-of-speech tagging or named entity recognition, we typically design “feature functions” which take the following parameters:

1. The entire sequence of words
2. The current index in that sequence
3. The predicted tags for the previous two words

The function then returns a set of features, usually binary, which might help predict the best tag for the given word.

The tag history is what makes sequence learning different from other types of machine learning. Without the tag history, we could come up with the features for each word (even if they use the surrounding words), and use something like logistic regression. In a sequence model, we can actually create features that use the predicted tag of the previous word.

Consider the use of the word “Brooklyn.” In isolation, we could assume it to mean the city, but it could be many other things e.g. Brooklyn Avenue, The Brooklyn Museum, etc. If we see “Brooklyn” and the last tag was HOUSE_NUMBER, it’s very likely to mean Brooklyn the street name. Similarly, if the last tag was HOUSE (our label for place/building name), it’s likely that we’re inside a venue name e.g. “The Brooklyn Museum.”

Features

The simplest and most predictive feature is usually the current word itself, but having the entire sequence means there can be bigram/trigram features, etc. This is especially helpful in a case like “Brooklyn Avenue” where knowing that the next word is “Avenue” may disambiguate words used out of their normal context, or help determine that a rare word is a street name. In a French address, knowing that the previous word was “Avenue” is equally helpful as in “Avenue des Champs-Élysées.”

Training the model for multiple languages entails a few more ambiguities. Take the word “de.” In Spanish it’s a preposition. If we’re lowercasing the training data on the way in, it could also be an abbreviation for Delaware (“wilmington de”) or Deutschland (“berlin de”). Again, knowing the contextual words/tags is quite helpful.

In libpostal, we make heavy use of the multilingual address dictionaries used above in normalization as well as place name dictionaries (aka gazetteers) compiled from GeoNames and OSM. We group known multiword phrases together so e.g. “New York City” will be treated as a single token. For each phrase, we store the set of tags it might refer to (“New York” can be a city or a state), and which one is most likely in the training data. Context features are still necessary though as many streets take their name from a proper place like “Pennsylvania Avenue,” “Calle Uruguay” or “Via Firenze.”

We also employ a common trick to capture patterns in numbers. Rather than consider each number as a separate word or token, we normalize all digits to an uppercase “D” (since we’re lowercasing, this doesn’t conflict with the letter “d”). This allows us to capture useful patterns in numbers and let them share statistical strength. Some examples might be “DDDDD” or “DDDDD-

DDDD" which are most likely US postal codes. This way we don't need many training examples of "90210" specifically, we just know it's a five digit number. GeoNames contains a world postal code data set, which is also used to identify potential valid postal codes. Some countries like South Africa use 4-digit postal codes, which can be confused for house numbers, and the GeoNames postal codes help disambiguate.

The learning algorithm

We use the **averaged perceptron**

(<http://www.cs.columbia.edu/%7Emcollins/papers/tagperc.pdf>) popularized by Michael Collins at Columbia, which achieves close to state-of-the-art accuracy while being much faster to train than fancier models like conditional random fields. On smaller training sets, the additional accuracy might be worth slower training times. On > 50M examples, training speed is non-negotiable.

The basic perceptron algorithm uses a simple error-driven learning procedure, meaning if the current weights are predicting the correct answer, they aren't modified. If the guess is wrong, then for each feature, one is added to the weight of the correct class and one is subtracted from the weight of the predicted/wrong class. The learning is done **online** (https://en.wikipedia.org/wiki/Online_algorithm), one example at a time. Since the weight updates are very sparse and occur only when the model makes a mistake, training is very fast.

In the averaged perceptron, the final weights are then averaged across all the iterations. Without averaging it's possible for the basic perceptron to spend so much of its time altering the weights to accommodate the few examples it gets wrong that it produces an unreasonable set of weights that don't generalize well to new examples (a.k.a. overfitting). In this way, averaging has a similar effect to regularization in other linear models. As in stochastic gradient descent, the training examples are randomly shuffled before each pass, and we make several passes over the entire training set.

Though quite simple, this method is **surprisingly competitive**

([http://aclweb.org/aclwiki/index.php?title=POS_Tagging_\(State_of_the_art\)](http://aclweb.org/aclwiki/index.php?title=POS_Tagging_(State_of_the_art))) in part-of-speech tagging, the existing NLP task that's closest to address parsing, and has by far the best speed/accuracy ratio of the bunch.

Evaluation

In part-of-speech tagging, simple per-token accuracy is the most intuitive metric for evaluating taggers and is used in most of the literature. For address parsing, since we'll want to use the parse results downstream as fields in normalization and search, a single mistake changes the

JSON we'll be constructing from the parse. Consider the following mistake:

```
Brooklyn/HOUSE Academy/HOUSE of/ROAD Music/HOUSE  
30/HOUSE_NUMBER Lafayette/ROAD Avenue/ROAD Brooklyn/CITY NY/STATE 11217/POSTCODE
```

In a full-text search engine like Elasticsearch, it might still work to search the name field with ["Brooklyn Academy", "Music"] plus the other fields and still get a correct result, but if we want to create a structured database from the parses or hash the fields and do a simple lookup, this parse is rendered essentially useless.

The evaluation metric we use is *full-parse* accuracy, meaning the fraction of addresses where the model labels every single token correctly.

On **held-out** ([https://en.wikipedia.org/wiki/Cross-validation_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics))) data (addresses not seen during training), the libpostal address parser currently gets **98.9% of full parses correct**. That's a single model across all the languages, variations, and field combinations we have in the OSM training set.

Future improvements

The astute reader will notice that there's still an open question here: how well does the synthesized training set approximate real geocoder input? While that's difficult to measure directly, most of the decisions in constructing the training set thus far have been made by examining patterns in real-world addresses extracted from the **Common Crawl** (<http://commoncrawl.org/>), as well as user queries contributed to the project by a production geocoder.

There's still room for improvement of course. Not every country is represented in the address formatting templates (though coverage continues to improve over time). Most notably, countries using the East Asian addressing system like China, Japan, and South Korea are difficult because the address format depends on which language/script is being used, necessitating some structural changes to the address-formatting repo. In OSM these addresses are not always split into components, possibly residing in the "addr:full" tag. However, since each language uses specific characters to delimit address components, it should be possible to parse the full addresses deterministically and use them as training examples.

The libpostal parser also doesn't yet support apartment/flat numbers as they're not included in most OSM addresses (or the address format templates for that matter). The parser typically labels them as part of the house number or street field. For geocoders, apartment numbers

aren't likely to turn up much as people tend to search at the level of the house/building number, but they may be unavoidable in batch geocoding. Supporting them would be relatively straightforward either by adding apartment or floor numbers to some of the training examples at random (without regard to whether those apartments actually exist in a particular building or not), or by parsing the "**addr:flats** (<http://wiki.openstreetmap.org/wiki/Key:addr:flats>)" key in OSM. The context phrases like "Apt." or "Flat" can be randomly sampled from any language in libpostal with a "unit_types" dictionary.

Conclusions

I'm hoping that libpostal will be the backbone for many great geocoders and apps in years to come. With that in mind, it's been designed to be:

1. International/multilingual
2. Technology and stack independent
3. Based on open data sets and fully open source

International by design, not as an afterthought

Almost every geocoder bakes in various myopic assumptions e.g. that addresses are only in the US, English, Latin script, the Global North, the bourgeoisie, etc.

Fully embracing L10N/I18N (localization/internationalization) means that there is no excuse for excluding people based on the languages they speak or the countries in which they live. An extra degree of rigor is required in recognizing and eliminating our own cultural biases.

There are of course always constraints on time and attention, so libpostal prioritizes languages in a simple, hopefully democratic way. Languages are added in priority order by the number of world addresses they cover, approximated by OpenStreetMap.

Usable on any platform

Libpostal is written in C mostly for reasons of portability. Almost every conceivable programming language can call into C code. There are already libpostal bindings for Python and NodeJS, and it's quite easy to write bindings for other languages.

Informed completely by open data

Libpostal makes use of several great open data sets to construct training examples for the address parser and language classifier:

- **OpenStreetMap** (<http://www.openstreetmap.org/>) is used extensively by libpostal to create millions of training examples of parsed addresses and language classifications.
- **GeoNames** (<http://www.geonames.org/>) is used by the address parser as a place name and postal code gazetteer, and will also be used for geographic name disambiguation in an upcoming release.
- **Quattroshapes** (<http://quattroshapes.com/>) and **Zetashapes** (<http://zetashapes.com/>) polygons are used in various places to add additional administrative and local boundary names to the parser training set. Zetashapes neighborhood polygons were particularly useful since neighborhoods are simple points in OSM.

All of the preprocessing code is open source, so researchers wanting to build their own models on top of open geo data sets are welcome to pursue it from any avenue (the puns just keep getting better) they choose.

The beauty of using these living, open, collaboratively edited data sets is that the models in libpostal can be updated and improved as the data sets improve. It also provides a great incentive for users of the library to support and contribute to open data.

Fin

You made it! The only thing left to do, if you haven't already, is check out libpostal on Github: <https://github.com/openvenues/libpostal> (<https://github.com/openvenues/libpostal>).

If you want to contribute and help improve libpostal, you don't have to know C, or any programming language at all for that matter. For non-technical folks, the easiest way to contribute is to check out our **language dictionaries** (<https://github.com/openvenues/libpostal/tree/master/resources/dictionaries>), which are simple text files that contain all the abbreviations and phrases libpostal recognizes. They affect both normalization and the parser. Find any language you speak (or add a directory if it's not listed) and edit away. Your work will automatically be incorporated into the next build.

Libpostal is already scheduled to be incorporated into at least 3 geocoding applications written in as many languages. If you're using it or considering it for your project/company, let us know.

Happy geocoding!

· 24 February 2016 ·



Al Barrentine is a Brooklyn-based data scientist passionate about open data, and believes high-quality data sets like Wikidata, OpenStreetMap, and Common Crawl are essential to creating a culture of reproducibility in machine learning.

© 2017 Mapzen

search (/tag/search) **geocoding** (/tag/geocoding)

Q Greenwich, london X

- Greenwich, Greater London, United Kingdom
- East Greenwich, Kent County, RI
- Greenwich, Fairfield County, CT
- Greenwich, Greater London, United Kingdom
- Old Greenwich, Greenwich, CT
- West Greenwich, Kent County, RI
- Greenwich, Lane Cove, Australia
- North Greenwich, Greater London, United Kingdom
- Greenwich, Washington County, NY
- Village of Greenwich, Greenwich, NY

Woolwich

Blackheath

Greenwich

Greenwich, Greater London, United Kingdom X

To clear up some terminology, we'll use the term "coarse geocoding" (for lack of something more descriptive) to refer to geocoding that doesn't involve streets. That is, inputs like:

- <https://mapzen.com/blog/geocoding-places/>

- 90210 (the postal code in Beverly Hills, not the TV show)

For clarity and when available, place names in this article will link to **Who's on First** (<https://whosonfirst.mapzen.com/>), the gazetteer being developed by Mapzen.

Coarse geocoding isn't quite as fraught with peril as address geocoding but it does present some interesting challenges. Let's get started!

Unambiguous Input

If you're lucky, the input supplied by the user can be geocoded to exactly one place.

Example: Truth or Consequences, NM

There is unambiguously a single place in New Mexico state named Truth or Consequences and it's a city.

- **Truth or Consequences, NM**
(<https://whosonfirst.mapzen.com/spelunker/id/85976585/>) (city)

Example: Bangkok, Thailand

There is a single city named Bangkok in Thailand

- **Bangkok, Thailand**
(<https://whosonfirst.mapzen.com/spelunker/id/102025263/>) (city)

Example: Maui, HI

Maui is a county, not a city, and no other place named 'Maui' in Hawaii exists.

- **Maui, HI** (<https://whosonfirst.mapzen.com/spelunker/id/102085577/>)
(county)

Example: Prince Edward Island

Most, but not all region names are unique. There is a single place named Prince Edward Island and it's a province in Canada. A geocoder should return:

- **Prince Edward Island**
(<https://whosonfirst.mapzen.com/spelunker/id/85682081/>) (province)

In all 4 cases above, a geocoder should return the 1 unambiguous result.

Alphanumeric Canadian postal codes

(https://en.wikipedia.org/wiki/Postal_codes_in_Canada) are unique in their format, so they should be treated homogeneously.

Example: K1A 0B1

A geocoder should return:

- **K1A 0B1 (a postal code in **Ottawa, Canada****
(<https://whosonfirst.mapzen.com/spelunker/id/101735873/>)

Ambiguous Inputs

More often than not, user inputs will be ambiguous, so business rules dictating result ordering should be applied. For instance, while not common, there are instances where the a city name appears multiple times within the same state.

Example: Tuckahoe, NY

A geocoder should return:

- **Tuckahoe, Suffolk County, NY**
(<https://whosonfirst.mapzen.com/spelunker/id/85979807/>)
- **Tuckahoe, Westchester County, NY**
(<https://whosonfirst.mapzen.com/spelunker/id/85978763/>)

Ordering of the two should be driven by an application's business rules. For example, the latter Tuckahoe is just to the north of New York City, so if the user has their map centered on New York City, it's entirely appropriate to return that one first. Alternately, business rules may want to return results in descending population order if the data is available.

Chance ordering should be avoided if possible since the user may not realize that an ambiguity exists.

Example: Las Vegas

Two US states can claim a place named Las Vegas, though most users are referring to the one in Nevada, not New Mexico, as the latter is much smaller. A geocoder should return:

- **Las Vegas, NV**
(<https://whosonfirst.mapzen.com/spelunker/id/85974801/>) (population 583,756)
- **Las Vegas, NM**
(<https://whosonfirst.mapzen.com/spelunker/id/85976683/>) (population 13,753)

Without state or business rules, users could be confused if a Las Vegas that they never heard of was returned before the Las Vegas that they were most likely expecting.

Some inputs could be interpreted ambiguously because it could be a city or county.

Example: Lancaster, PA

Lancaster is both a city and county in the state of Pennsylvania. A geocoder should return:

- **Lancaster, PA**
(<https://whosonfirst.mapzen.com/spelunker/id/101718643/>) (city)
- **Lancaster, PA**
(<https://whosonfirst.mapzen.com/spelunker/id/102081377/>) (county)

While counties are important governmentally, users are normally looking for the city when the name could be interpreted either way. In this case, it's safe to return the city first and the county second, but your business rules may dictate otherwise.

Some inputs are wildly popular place names and may appear anywhere in the administrative hierarchy.

Example: Luxembourg

A geocoder should return:

- **Luxembourg** (<https://whosonfirst.mapzen.com/spelunker/id/85633275/>) (country)
- **Luxembourg** (<https://whosonfirst.mapzen.com/spelunker/id/101751765/>) (city)
- **Luxembourg** (<https://whosonfirst.mapzen.com/spelunker/id/85673875/>) (region)

Countries with less-than-imaginative postal code formats can result in ambiguities, too. For instance, the US 5-digit postal code format is used by a number of countries. Postal codes may not be included in the gazetteer serving as the basis for your source data and may need to be augmented from additional sources.

Example: 90210

This specific postal code appears in a number of countries, such as the United States, Mexico, and Thailand. A geocoder should return results based on what data is available.

It's even possible for what appears to be a region and country to be interpreted as a city and state.

Example: Ontario, CA

In this case, both Ontario, California and Ontario, Canada are legitimate interpretations of the users input.

A geocoder should return:

- **Ontario, Canada** (<https://whosonfirst.mapzen.com/spelunker/id/85682057/>) (province)
- **Ontario, California** (<https://whosonfirst.mapzen.com/spelunker/id/85923933/>) (city)

Anomalous Inputs

In a perfect world, users would only enter inputs that make sense. We don't live in a perfect world so geocoders have to deal with inputs that make little (if any) sense.

Mismatched cities, states, and countries

One common type of anomaly is a mismatched city and state combination.

Example: Hilton Head, North Carolina

The city of **Hilton Head**

(<https://whosonfirst.mapzen.com/spelunker/id/101720697/>) is actually in South Carolina but users seem to confuse the states quite a bit.

A geocoder should return:

- **Hilton Head**
(<https://whosonfirst.mapzen.com/spelunker/id/101720697/>), SC

In the same vein, cities in regions that border other countries can be confused by users.

Example: Strasbourg, Germany

Strasbourg (<https://whosonfirst.mapzen.com/spelunker/id/101751113/>) is in the Alsace-Champagne-Ardenne-Lorraine region of France but within mere kilometers of Germany. A glance at a map in that general area of France shows many town and city names that could easily be mistaken for German by users unfamiliar with the geographical nuances.

A geocoder should return:

- **Strasbourg** (<https://whosonfirst.mapzen.com/spelunker/id/101751113/>), FR

Changing country borders can lead to users entering city and country combinations that aren't correct.

Example: Juba, Sudan

South Sudan officially split from Sudan in 2011, naming Juba it's capital.

A geocoder should return:

- Juba, SS

When inputs aren't concordant, then they're not easy to geocode, so it's time to apply editorial and/or algorithmic business rules. Editorial mechanisms can be a quick fix but require constant maintenance and log file analysis. If your gazetteer has adjacency information or bounding polygons, it's possible to correct mistakes such as entering a city with a state right next to it, as in the 'Hilton Head, NC' example.

Inputs that just don't make sense

Not all anomalous inputs can be solved editorially or algorithmically.

Example: Las Vegas, Maine

There's no neighborhood, city, or county named 'Las Vegas' in the state of Maine

Regardless of how incongruous an input is, it's what the user entered it so a geocoder should try to make some basic sense of it. The input may not even be a mismatch but something spelled so horrendously incorrect that no algorithmic spell checker in the world can fix it. In this case, there are two general approaches a geocoder can take:

- Return **'Maine (<https://whosonfirst.mapzen.com/spelunker/id/85688769/>)'** (the state). In this case, the user is clearly looking for a place in Maine, our geocoder just isn't sure what, so it's safe to return 'Maine'.
- Return **'Las Vegas, NV (<https://whosonfirst.mapzen.com/spelunker/id/85974801/>)'**. 'Las Vegas' is a pretty big city, so it's possible that the user was just confused on which state Las Vegas is in.

A combination of the two approaches is achievable with available population or popularity data to classify the city portion as big enough to pass a scoring threshold. In either approach, efforts should be made to inform the calling application that the granularities of information entered versus returned were different so that actions can be taken to correct the error.

Diacritical Marks

Efforts should be made to accommodate users who may not be aware of place names containing **diacritical marks** (<https://en.wikipedia.org/wiki/Diacritic>).

Example: Munster, Germany

A geocoder should return:

- **Münster, Germany**
(<https://whosonfirst.mapzen.com/spelunker/id/101913895/>)

While certainly important to pronunciation, it's unreasonable to expect users unfamiliar with a language to enter diacritical marks correctly.

Misspellings

Some place names, like "New York" and "London" are arguably easy to spell (at least if you're a native speaker), but others aren't. I consider myself an excellent speller in English but struggle to remember how many P's and S's are in **Parsippany**

(<https://whosonfirst.mapzen.com/spelunker/id/404477665/>) or which **Pittsburgh**

(<https://whosonfirst.mapzen.com/spelunker/id/101718805/>)'s end in H's. Woe be to the non-native English speaker trying to remember the rules (and exceptions to those rules).

Autocomplete engines can help a bit for patient users, but a spelling-correction library should be included in any geocoder. There are a number of open-source libraries for this task from the simple n-gram approach up to more complex approaches like **Hunspell** (<http://hunspell.github.io/>).

What's Next

Parts 1 (<https://mapzen.com/blog/meaningful-geocoding-address-search-the-two-core-principles-of-geocoding>) and 2 of this series covered how to geocode streets and administrative regions such as city, states, countries, and postal codes. It's not enough to just

return results, that which calls the geocoder must also be given information on how the results should be interpreted. In part 3, we'll discuss the different types of information that applications would be interested in.

If cracking the code of how we organize the world sounds interesting to you, we'd love to work with you. Our project is 100% open, so it would be great to have you as an **open source contributor** (<https://github.com/pelias/pelias>), not to mention we're hiring for another person to **join our Search team** (<https://mapzen.com/jobs/search-engineer-node/>) to work on geocoding full time.

· 01 March 2016 ·



Stephen Hess

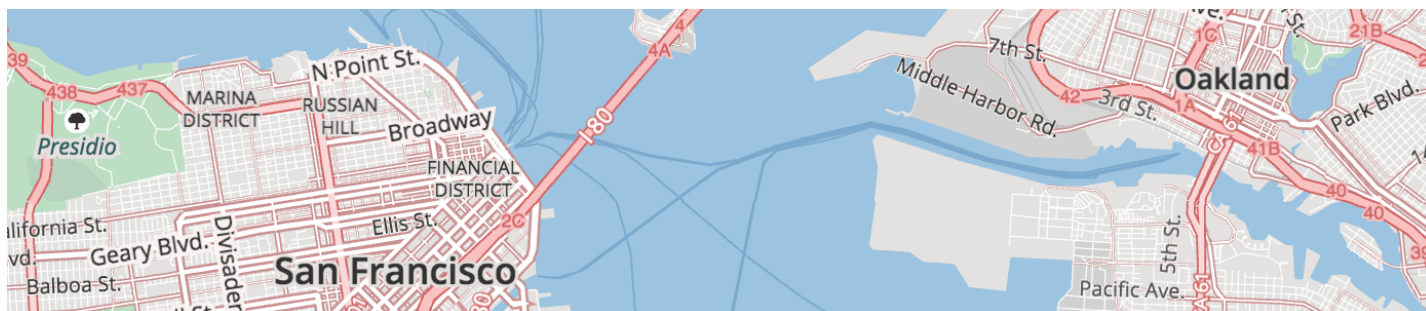
Stephen works on geocoding exclusively as a means to fund his passion for designing and building wooden frames for old maps.

© 2017 Mapzen

Mapzen's Vector Tile service — better, stronger, faster

vector-tiles ([/tag/vector-tiles](#)) **data** ([/tag/data](#)) **documentation**

([/tag/documentation](#)) **osm** ([/tag/osm](#)) **whosonfirst** ([/tag/whosonfirst](#))



A lot has **changed** (<https://github.com/mapzen/vector-datasource/blob/v0.7.0/CHANGELOG.md>) since we **launched** (<https://mapzen.com/blog/look-upon-our-squares-of-math-in-three-dimensions>) the Mapzen Vector Tiles service last year, including **updated documentation** (<https://mapzen.com/documentation/vector-tiles/>). I'd like to give you a quick update on recent changes and what's about to land.

Not familiar with vector tiles? The **Mapzen vector tile service** (<https://mapzen.com/projects/vector-tiles>) provides worldwide basemap coverage sourced from **OpenStreetMap** (<https://www.openstreetmap.org>) and other open data projects, updated daily as a free & shared service.

Data is organized into several thematic layers, e.g. `buildings`, `pois`, and `water`. Each layer includes a simplified view of OpenStreetMap data for easier consumption, with common tags often condensed into a single `kind` property.

As we work towards a v1.0 release this year we'd love to hear your feedback on the Mapzen Vector Tile service – please say hello@mapzen.com (<mailto:hello@mapzen.com>). (If you are at the **State of the Map US in Seattle in July 23-25th** (<https://openstreetmap.us/2016/02/sotmus-2016/>), we can talk about it in person!)

Updated documentation

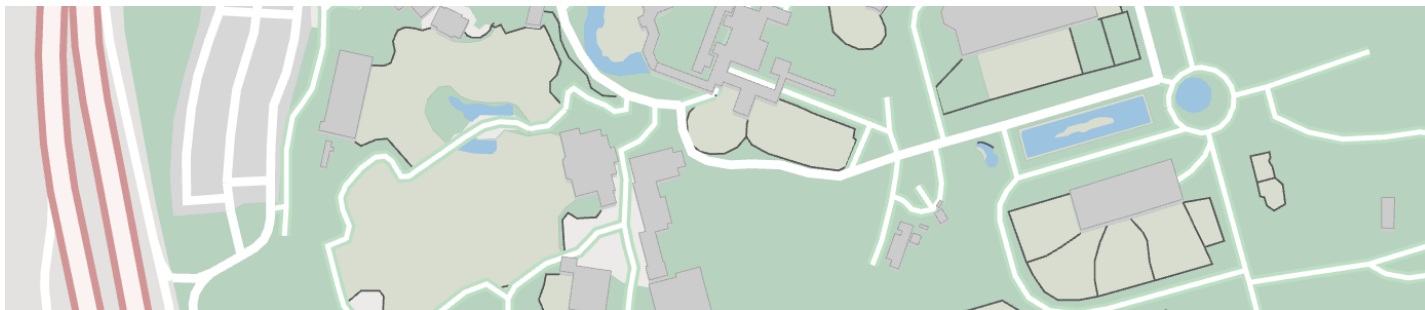
We've published updated **documentation** (<https://mapzen.com/documentation/vector-tiles/>), including sections for data sources and attribution, name localization, geometry types, data updates, general changelog, and full layer reference with map previews and lists of property keys and their values.

Faster, fresher tiles

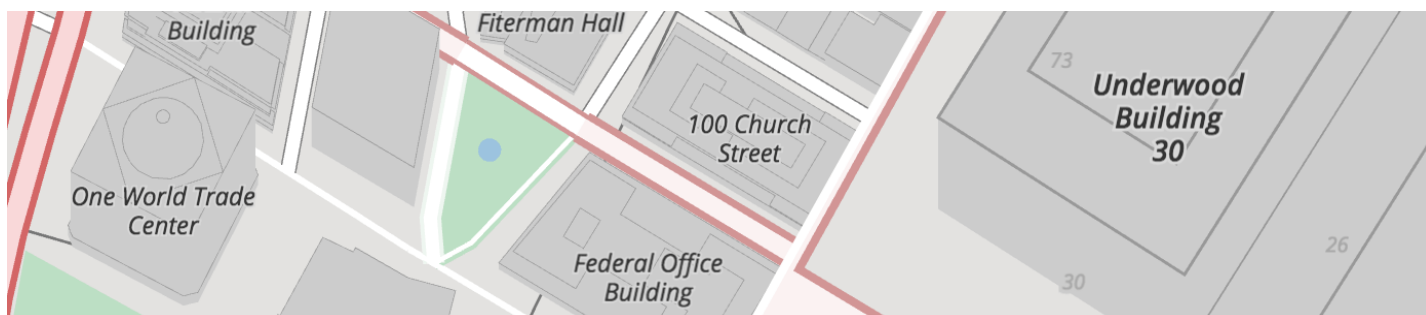
Starting last week, customers in the western United States, Europe, and Asia should see faster tile load times, and data updates from OpenStreetMap now appear in Mapzen tiles on a more frequent basis.

New tile data

Now at version v0.7, we've had 6 major updates to Mapzen Vector Tiles since we released the service last year. Here are some highlights!



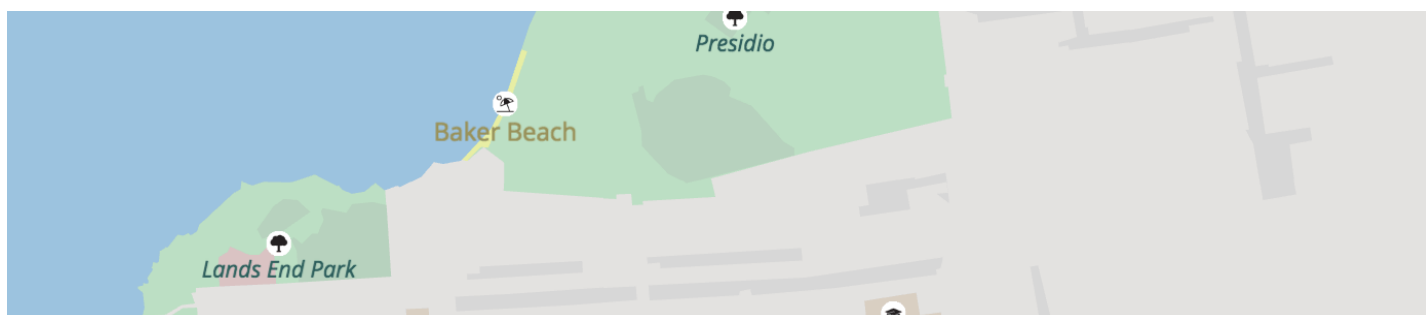
Boundaries: Fences, city walls, and other barriers have been added to the `boundaries` layer. (above) the San Francisco Zoo.



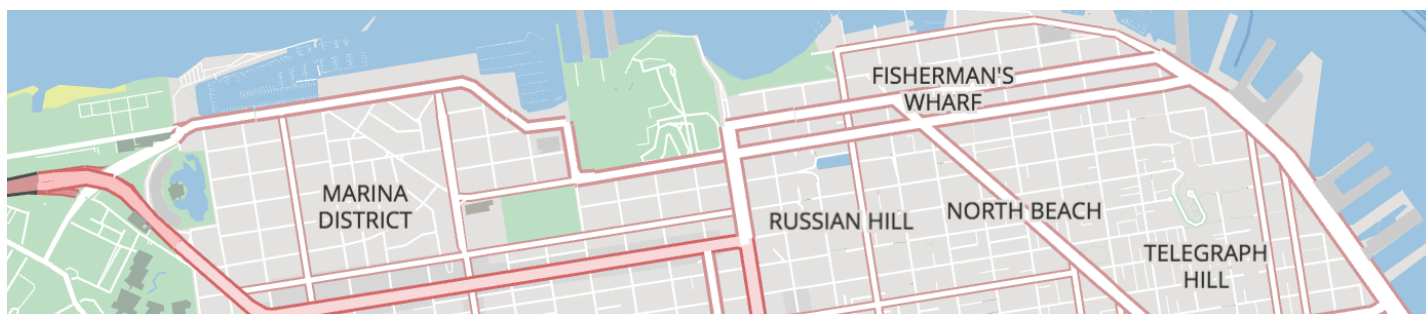
Buildings: Name label positions (deduplicated across tiles) and addresses have been added to the `buildings` layer.



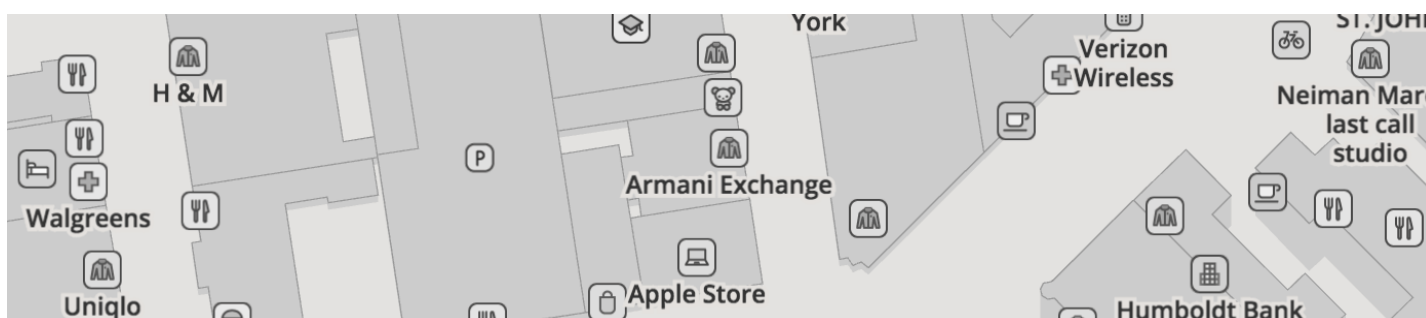
Mapzen calculates a `landuse_kind` values for buildings by intercutting them with the `landuse` layer to determine if they are over parks, hospitals, universities or other landuse features. Use this property to modify the visual appearance of buildings over these features. For instance, light grey buildings look great in general, but aren't legible over most landuse colors unless they are darkened (or colorized to match landuse styling).



Landuse: The `landuse` layer also includes new label placement points (deduplicated across tiles), and many more landuse kinds including beaches.



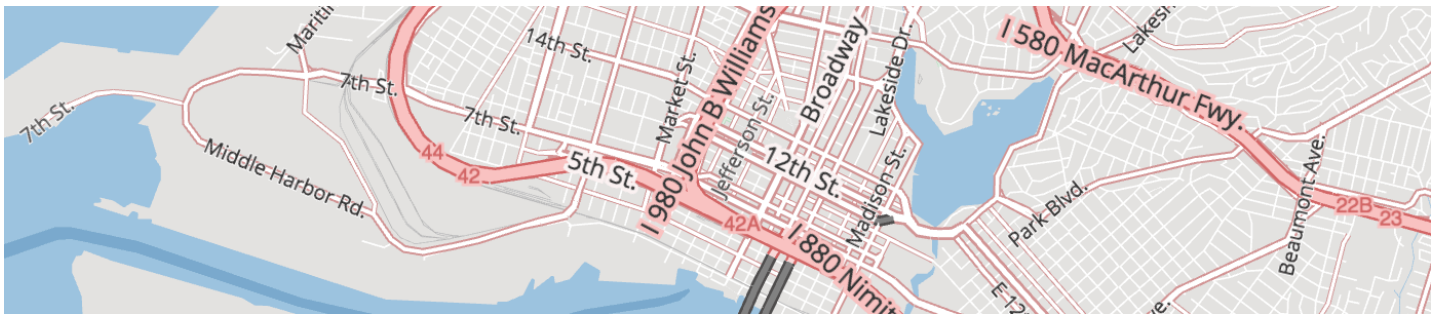
Neighbourhoods: Starting at zoom 12 `neighbourhood` and `macrohood` features are added from **Who's On First** (<http://whosonfirst.mapzen.com>) to the `places` layer.



POIs: We now support over 200 points of interest in the `pois` layer, and most have custom sprite icons.

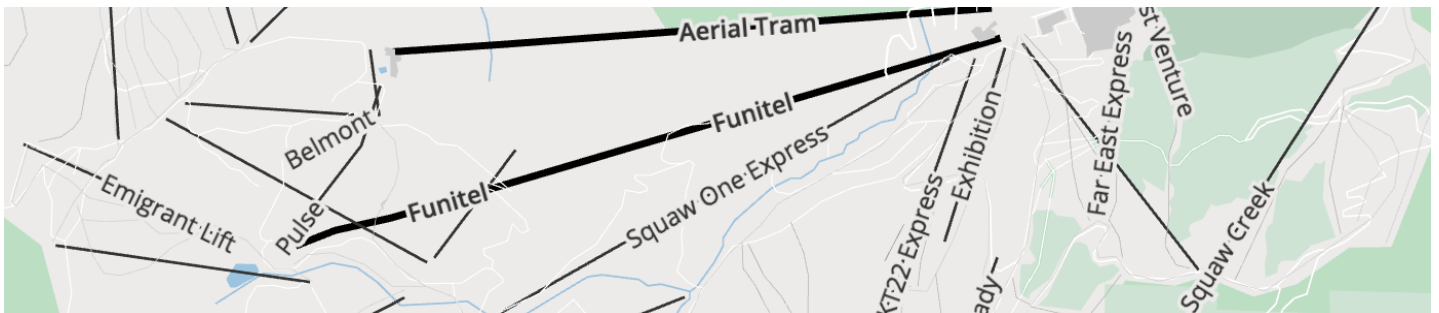
Different kinds of POIs are included in tiles based on zoom, but we promote some features up a few zooms in each kind based on their building (or landuse) areas. For instance, a big university might be visible at zoom 12, but a small (or no area) university might only be visible at zoom 15.

Transit stations include a `tile_kind_rank` (as detailed in Matt's **station relations** (<https://mapzen.com/blog/station-relations>) post) and indicate which transit routes service the station.



Roads: We are including more transportation features in the `roads` layer, and road names are now abbreviated. Directionals like `North` and `Northeast` are replaced with `N` and `NE`, and common street suffixes like `Avenue` and `Street` change to `Ave.` and `St.`. For low- and mid-zooms, some properties, like `is_tunnel`, are dropped.

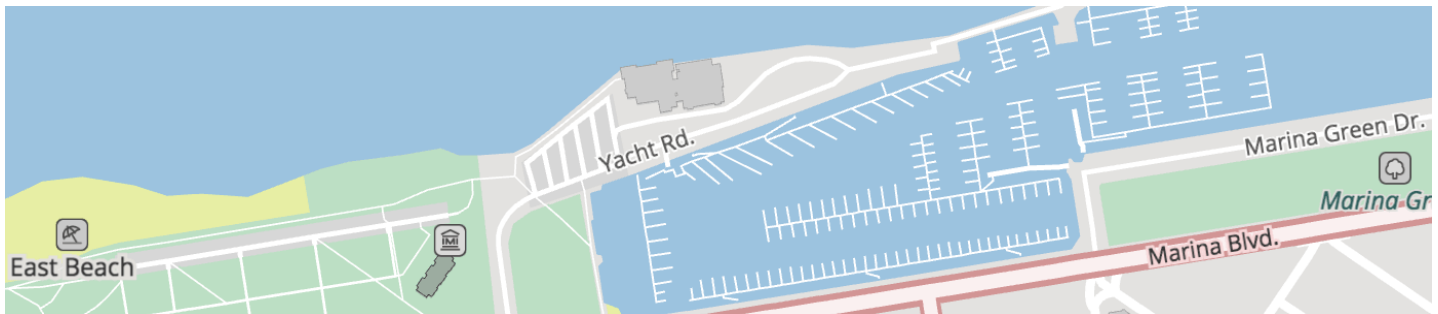
Mapzen calculates the `landuse_kind` values for roads by intercutting them with the `landuse` layer to determine if a road segment is over parks, hospitals, universities or other landuse features. While light grey minor roads look great in general, they aren't legible over most landuse colors unless they are darkened. As with `buildings`, you can use this property to modify the visual appearance of roads over these features.



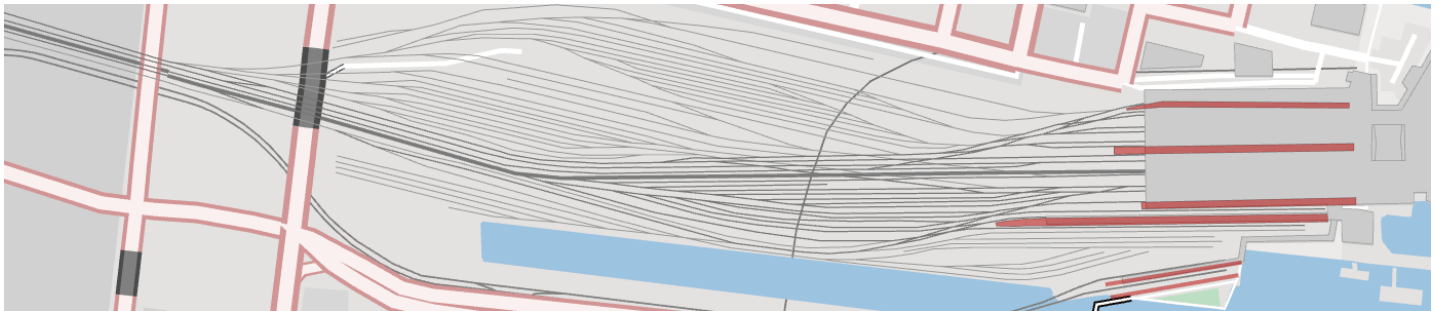
Aerialways Gondola, cable car, chair lifts, and other aerialways were added a while back.



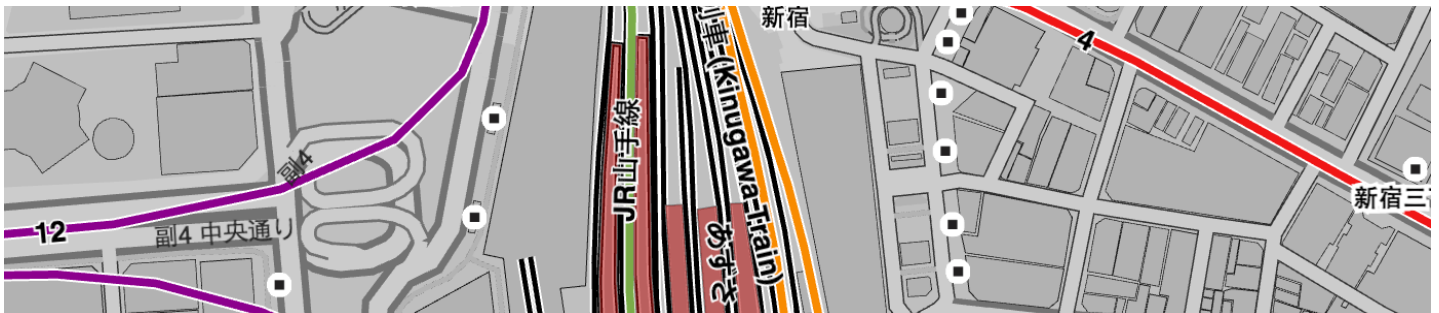
Ski pistes: More interested in hitting the slopes? Head for the black diamonds.



Piers start showing up at zoom 13+ with a kind value of `pier`.



Railroads: New details for yard, transit, and spur.



Transit: The `transit` layer includes line features from OpenStreetMap, which start appearing at zoom 6+ for trains and railways. Then subway, light rail, and tram are added at zoom 10+. Platform polygons are added at zoom 14+.



Water: The `water` layer includes new label placement points (deduplicated across tiles).

Upcoming changes that may affect your map

Mapzen plans to release v0.8 of the Vector Tiles Service on March 8th. This milestone has been focused on improving server and client performance. Some of those changes went live last week, and the final data changes are being staged now.

1. Significantly more points of interest (POIs) and boundaries will appear at zooms 16 and 17 than before.
2. New `sort_key` values will be provided to order `landuse` and `road` features that are incompatible with the earlier `sort_key` values.

Overstuffing zoom 16 & 17 tiles

The upcoming v0.8 vector tiles release will include data changes to zoom 16 tiles that will significantly increase the quantity of `pois` layer features and includes a minor change to the `boundaries` layer, which affects the overall balance of the map. This gives map designers more control over what shows up when, and it improves client and server performance.

Features previously limited to zoom 17 and 18 tiles only (including hotels, ATMs, bus stops, and parking lots) will now be visible starting at zoom 16. Zoom 17 tiles will be 99% the same as before (with a couple of zoom 18 features added in), and zoom 18 tiles will be exactly the same as before.

If you are **not** currently showing `pois` or `fence` `boundaries` on any of your maps, this change doesn't affect you. If you **are**, then it's a **16 line fix**

(<https://gist.github.com/nvkelso/2b8d50a948c5abc749aa>) in a Tangram scene file to do the filtering client side to continue the old customer experience. Paired with this change, we will provide the recommended `min_zoom` on all features in the `pois` layer so you'll still know the recommended zoom to show each feature.

This change allows client mapping libraries like Tangram to stop requesting vector tiles after zoom 16, improving client-side map drawing performance at zooms 17+. Higher zoom level tiles will continue to be supported though.

Changes to sort_key values

The v0.8 release will include data changes to `sort_key` values for `landuse` and `roads` features at all zoom levels (and implied `sort_key` values for features in other layers). These new `sort_key` values are incompatible with earlier `sort_key` values, particularly if you've manually set a z-index order for features in the `water`, `boundaries`, or other layers that depends on `landuse` and `roads` feature orders.

Up until now, we have packed landuse features into 5 non-unique `sort_key` values. While this worked for 2D (top down) map views, it sometimes introduced z-fighting in tilted 3D views (like parking lots over shopping mall landuse polygons). Similarly, road tunnels sometimes sorted below water (instead of being transparent above water), and we didn't allow enough gaps in the range to insert other features like buildings between tunnels and surface streets. The new `sort_key` values provide over 10x the range as before, ensuring that each feature class (or kind) has its own GL turf.

If you are **not** currently ordering landuse or roads using Mapzen's `sort_key` convenience values, this change doesn't affect you. If you **are**, then please contact us for a migration strategy.

Again, as we work towards a v1.0 release we'd love to hear your feedback about Mapzen's Vector Tiles service. Please say **hello@mapzen.com** (<mailto:hello@mapzen.com>)!

· 02 March 2016 ·



Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.



Rob Marianski

Software engineer working on cutting tiles and infrastructure. Functional programming enthusiast.



Matt Amos

OpenStreetMap developer-contributor and chilli enthusiast. Writes vector tile and other data-mastication tools at Mapzen.

© 2017 Mapzen

Give your vector tile life

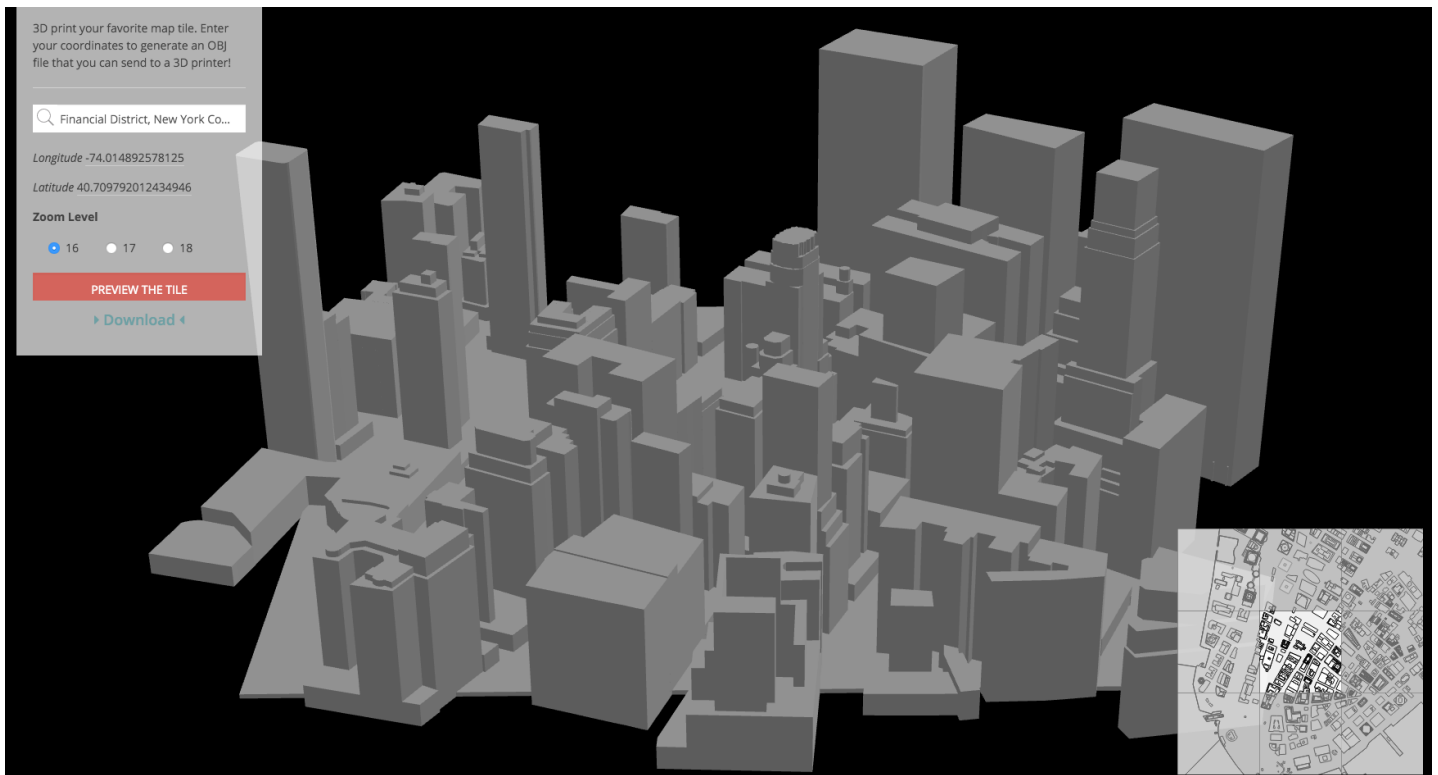
Vector Tile ♥ Digital Fabrication

I really like making physical objects, but I am clumsy at handiwork. Do you know a person who just can't cut a straight line along the guide? That's me. I can't, I just can't. This is why digital fabrication has very special place in my heart. Laser cutters, 3D printers, computerized embroidery machines, CNCs... they make me feel like Iron Woman.

Mapzen's vector tile page (<https://mapzen.com/projects/vector-tiles/#why-vector-tiles?>) explains what a vector tile is, and how we can benefit from it compared to raster tiles, but there is one thing that I'd like to add. Vector tiles are ✨awesome✨ for digital fabrication since most digital fabrication tools accept vector graphic formats as an input. I ♥ maps, I ♥ fabrication, and I have vector tiles to use, so let's get to work.

Here it is

Since the vector tiles contain three-dimensional information, I decided to fabricate them with a 3D printer. To prepare the tiles for the printer, we first created a tool called the **Tile Exporter** (<http://hanbyul-here.github.io/tile-exporter/>). It grabs a **Mapzen vector tile** (<https://mapzen.com/projects/vector-tiles>), offers you 3d preview in your browser, and then creates an .OBJ file of the scene that you can download.



You can use the search box in the left top corner (powered by **Mapzen search** (<https://mapzen.com/projects/search>)) to find an address or a place, and choose your zoom level. The bigger the zoom level, the smaller the area and greater the detail. Hit 'Preview the tile' button after your selection is done. Didn't get the exact point you wanted? Use the navigation maps at the bottom right — you can move in 8 directions! (The navigation map even uses the same data that eventually generates the tiles - the freedom of vectors!)

The tile exporter gets the `buildings`, `earth`, `water`, `landuse` layers of a tile. Learn more about layers in tiles at the **Mapzen Vector Tile documentation** (<https://mapzen.com/documentation/vector-tiles/layers/>). There is also a good read from **previous blog post** (<https://mapzen.com/blog/getting-crafty/#wild-tile>) about how Open Street Map data is represented in a tile.

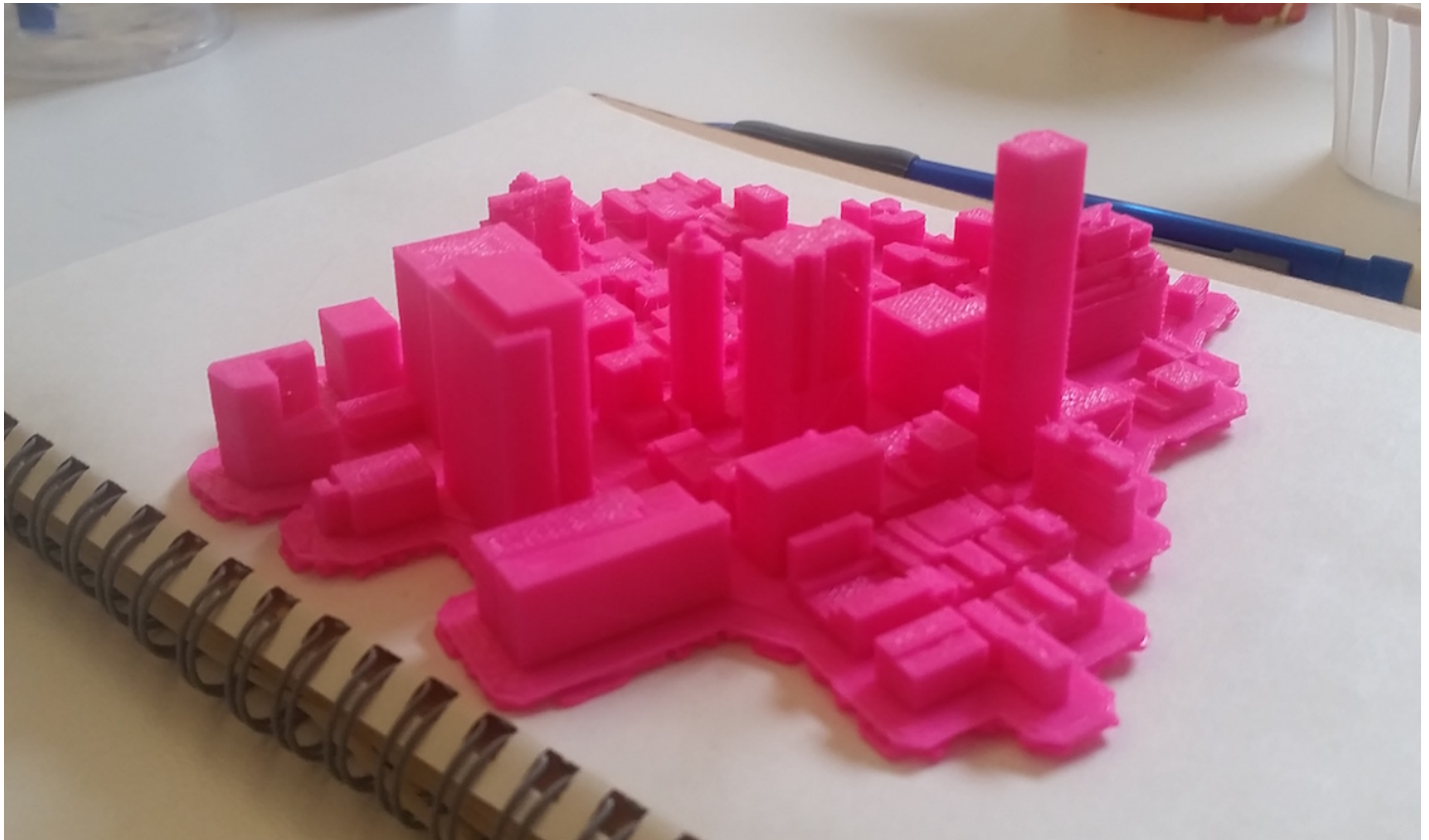
Quilt of open source

This exporter is a true quilt of many open source projects. First, geojson from a **vector tile** (<https://mapzen.com/projects/vector-tiles>) is passed to **D3** (<https://d3js.org/>) to generate the SVG. This SVG is then processed into a form suitable for **Three** (<http://threejs.org>) with **d3-three js** (<https://github.com/asutherland/d3-threeD>), which lets Three **smartly extrude** (http://threejs.org/examples/#webgl_geometry_extrude_shapes2) the processed SVG into a 3D shape based on its `height` information. Three also offers an **OBJ exporter**

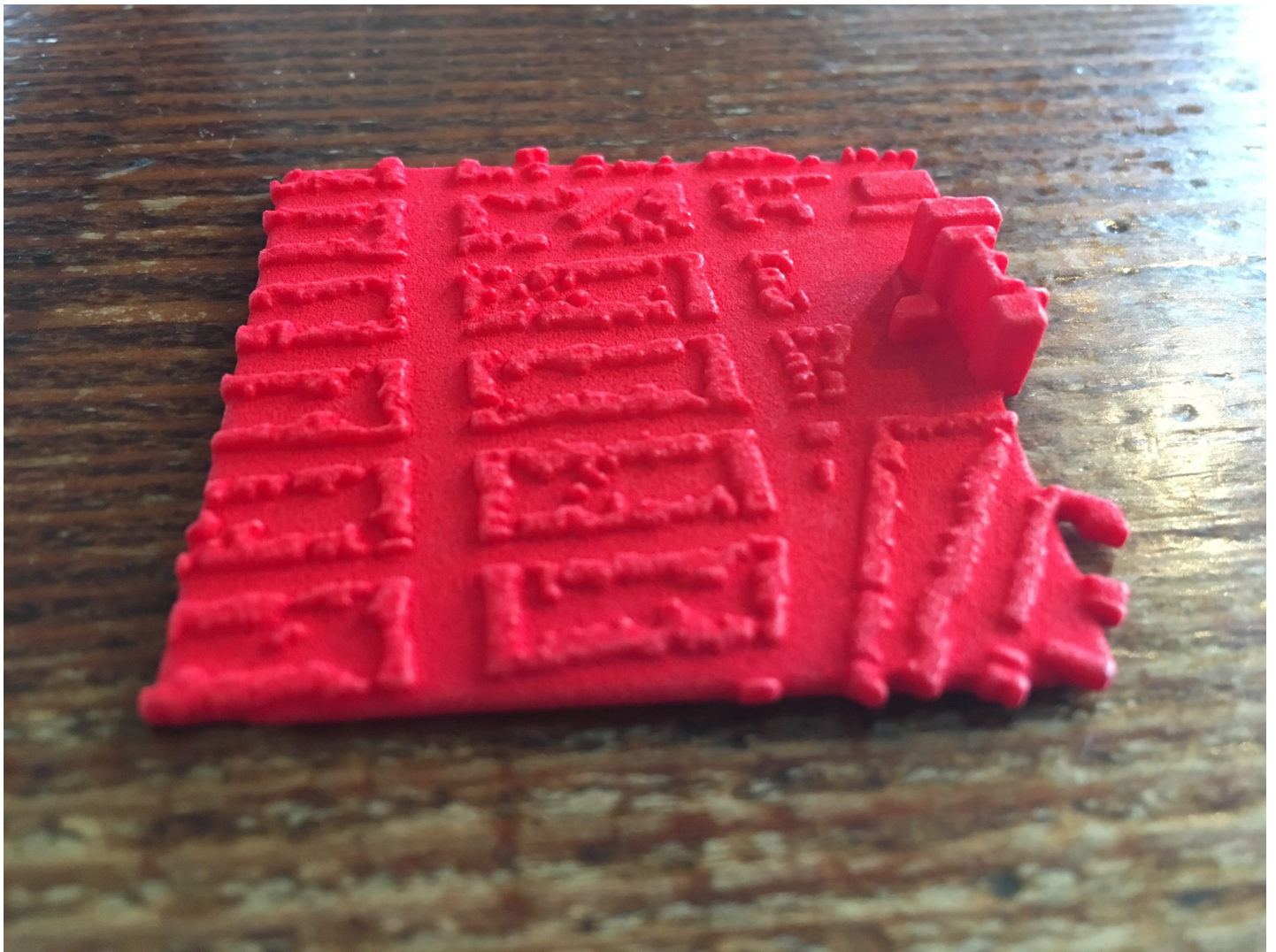
(http://threejs.org/examples/#webgl_exporter_obj), which makes it really easy to grab the OBJ file from the scene - and it's an OBJ that the 3D printer wants. The tile exporter can only exist thanks to all of these awesome open source tools working together.

· ° ✧ · ° ✧ *: **Let's** *: · ° ✧ · ° ✧

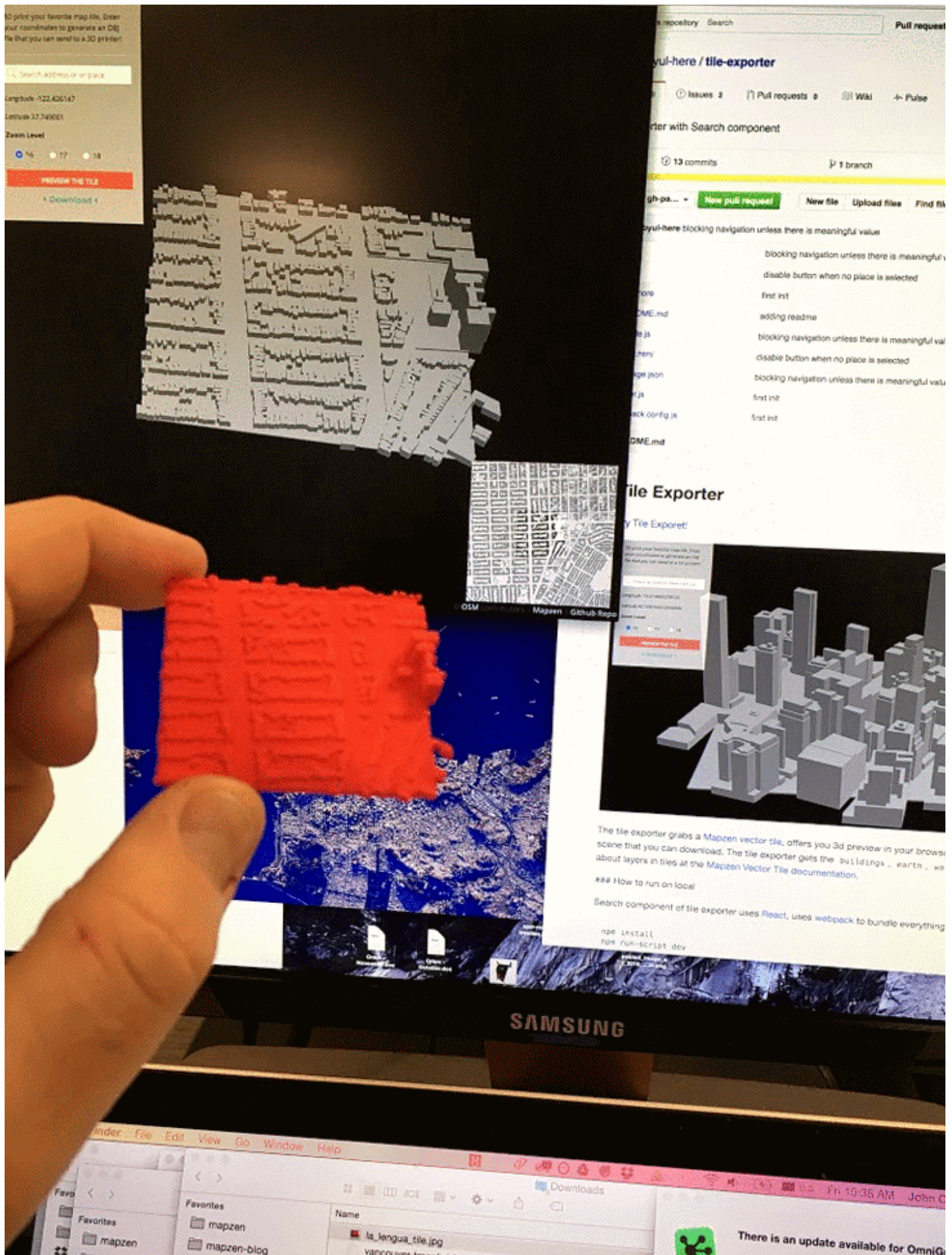
We made this tile of southern Manhattan on our 3D printer (**MakerBot** (<http://www.makerbot.com/>)) in our New York office.



We made this OBJ file of the heart of **La Lengua** (<https://whosonfirst.mapzen.com/spelunker/id/102112179/>) in San Francisco and sent it to Shapeways (<https://www.shapeways.com/>)!



Getting meta: a GIF of a 3D-printed tile in front of an OBJ file above an SVG of a vector tile:



Don't see your favorite building in a tile? It means that OpenStreetMap doesn't have information about it yet, so please **contribute** (<http://learnosm.org/en/>)! And **let us know what tiles you make!** (<https://twitter.com/intent/tweet?text=@mapzen%20I%20made%20a%203D%20tile!>)

· 04 March 2016 ·



Hanbyul Jo

Hanbyul does front-end that makes maps and candies.



John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

Open Data hackathons and open transit feeds

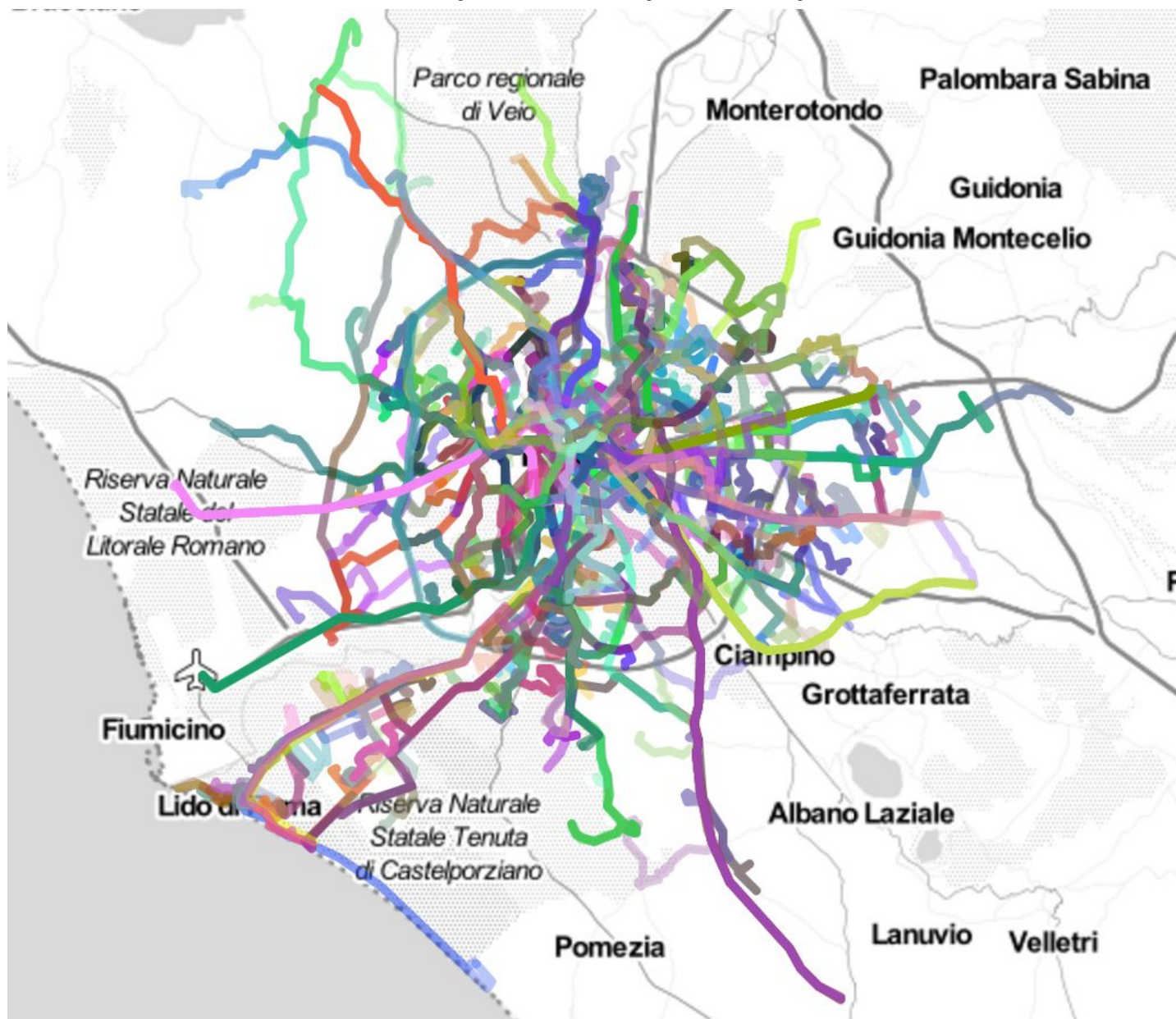
No weekend plans? Want to meet some people and play with bus/train/tram data? If so, you're in luck: This Saturday is the **International Open Data Day** (<http://opendataday.org/>) hackathon, and both Saturday and Sunday are **CodeAcross** (<https://www.codeforamerica.org/events/codeacross-2016/>) hackathon days. **Transitland** (<https://transit.land>)—an open-transit-data service sponsored by Mapzen and supported by an increasing number of contributors—will be there, too.

To preview what you can create this weekend, and any other time, we'd like to highlight a few of the ways data enthusiasts, transit agencies, and app developers are contributing to Transitland in Rome, Chicago, Buenos Aires, and Washington, D.C.

Rome

Andrea Borruso added feeds to Transitland for several Italian transit agencies, including **Roma Servizi per la Mobilità s.r.l.** (<https://transit.land/feed-registry/operators/o-sr2-romaserviziperlamobilitasrl>), and was kind enough to write a blog post in Italian about Transitland (<http://blog.spaziogis.it/2016/03/02/transiland-per-mettere-insieme-e-dare-vita-ai-dati-sui-trasporti/>). Grazie, Andrea!

Here's what Rome's transit network looks like using the **Transitland Playground** (<https://transit.land/playground/>) data explorer and downloader:



If you happen to be in Italy, there are several Open Data Day hackathons – check out **Andrea's Twitter feed** (<https://twitter.com/aborruso/>) for a good overview of what's going on.

Looking for GeoJSON for the route geometries of your favorite transit system? It's accessible through the **Transitland Datastore API** (<https://transit.land/how-it-works/#slide-3>) like so:

```
https://transit.land/api/v1/routes.geojson?operatedBy=[operator_onestop_id]
```

This can be imported and displayed in any application or website that understands GeoJSON. **In the case of Rome** (<https://transit.land/api/v1/routes.geojson?operatedBy=o-sr2-romaserviziperlamobilita>), it's

<https://transit.land/api/v1/routes.geojson?operatedBy=o-sr2-romaserviziperlamobilitasrl> (7 MB)

Want to look up the Onestop ID for a different operator? Browse the **Transitland Feed Registry** (<https://transit.land/feed-registry>) or query the Datastore API like so:

<https://transit.land/api/v1/operators>

Chicago

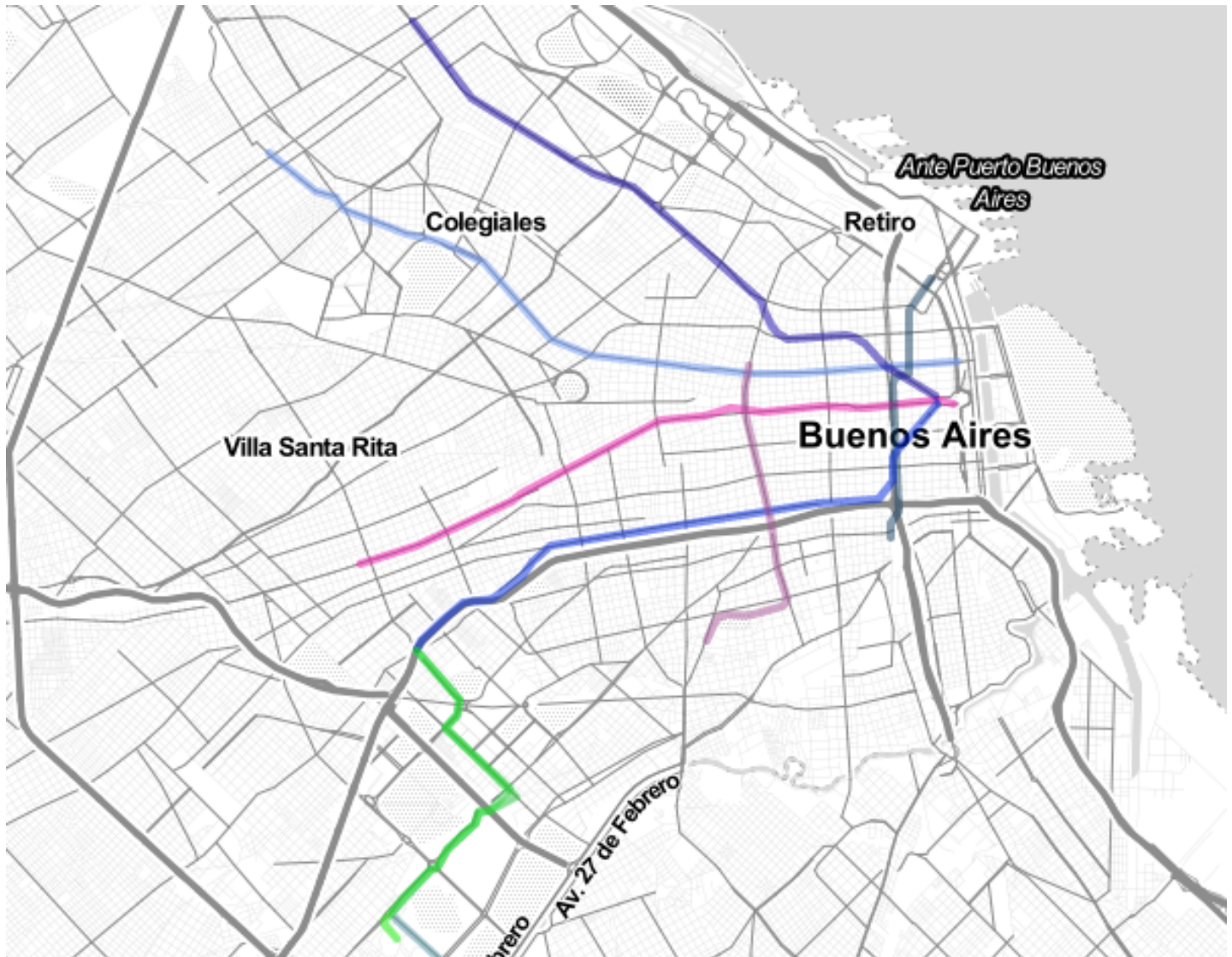
Twenty-five or six to four (<http://www.songfacts.com/detail.php?id=1197>) aside, the **Chicago Transit Authority's GTFS feed** (<https://transit.land/feed-registry/operators/o-dp3-chicagotransitauthority>) is now in Transitland's Feed Registry. **Daniel Burhham** (https://en.wikipedia.org/wiki/Daniel_Burnham)'s mark on the city is clear.



CTA GTFS GeoJSON (12.8MB) (<https://transit.land/api/v1/routes.geojson?operatedBy=odp3-chicagotransitauthority>)

Buenos Aires

The Buenos Aires subway, **Subterráneos de Buenos Aires (SUBTE)** (<https://transit.land/feed-registry/operators/o-69y7-sbase>), has eight lines.



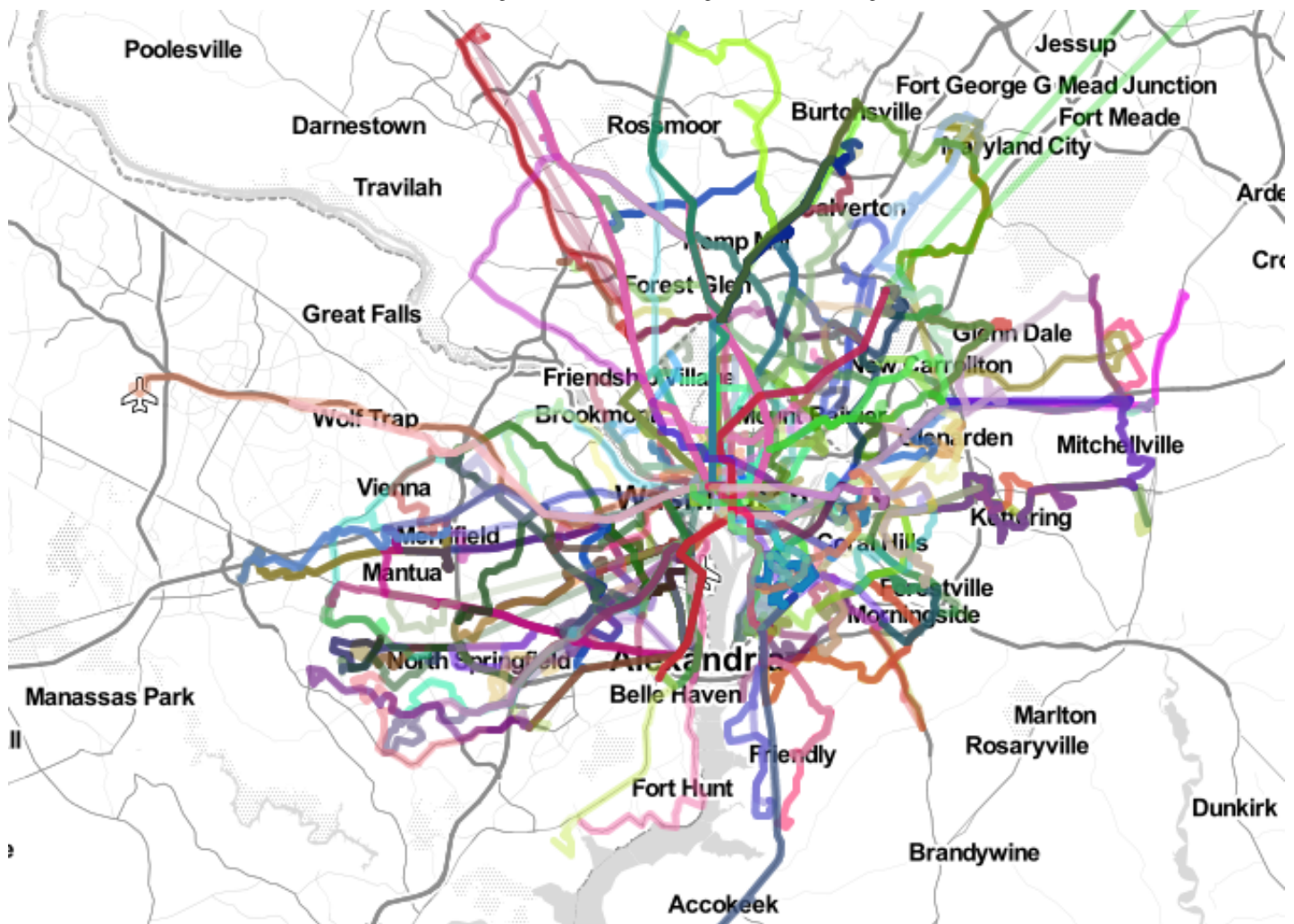
More information about the *Subterráneos* routes can be easily viewed **using the Datastore API** (<https://transit.land/api/v1/routes?operatedBy=o-69y7-sbase>) and **mapped as GeoJSON** (<https://transit.land/api/v1/routes.geojson?operatedBy=o-69y7-sbase>) (38K)

```
{
  "routes": [
    {
      "identifiers": [
        "gtfs://f-69y7-recursosdatabuenosairesgobar/r/PM-Savio"
      ],
      "imported_from_feed_onestop_ids": [
        "f-69y7-recursosdatabuenosairesgobar"
      ],
      "imported_from_feed_version_sha1s": [
        "4e07fa71ddcc48667af3ccf61d49f85056e1b305"
      ],
      "created_or_updated_in_changeset_id": 605,
      "onestop_id": "r-69y6v-pm~s",
      "name": "PM-S",
      "vehicle_type": "tram",
      "geometry": {
        ...
      }
    }
  ]
}
```

The GeoJSON you get from the Transitland API can be displayed in many different web apps, including geojson.io! **Click here to see SUBTE routes using geojson.io** (<http://geojson.io/#data=data:text/x-url,https%3A%2F%2Ftransit.land%2Fapi%2Fv1%2Froutes.geojson%3FoperatedBy%3Do-69y7-sbase>).

Washington D.C.

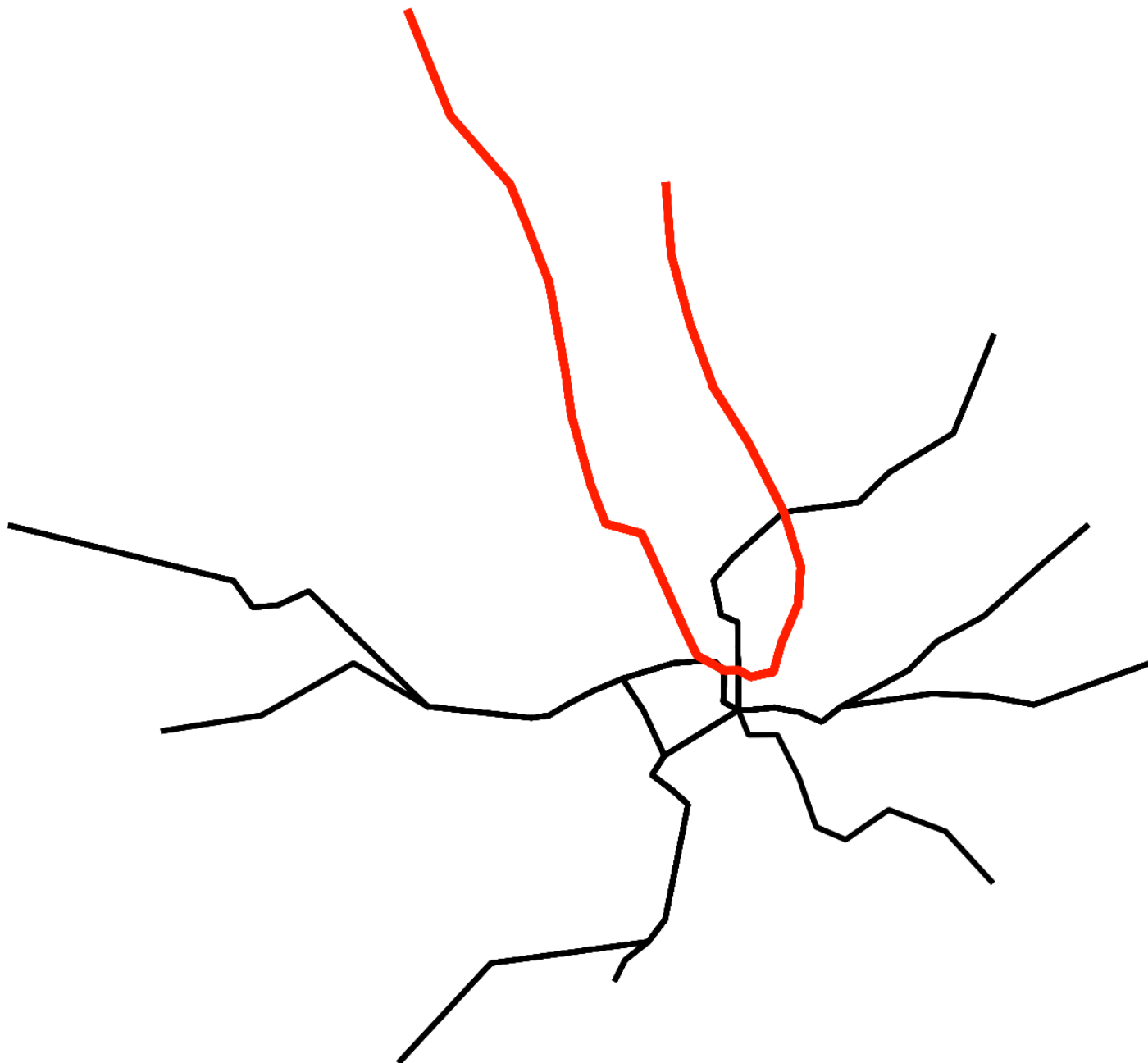
I pity the fool who takes a cab in DC (https://en.wikipedia.org/wiki/D.C._Cab)—you should be **riding WMATA when in the District of Columbia** (<https://transit.land/feed-registry/operators/o-dqc-met>).



Lots of lines! But through the power of the Transitland API, you can pull out just the subways...

https://transit.land/api/v1/routes.geojson?vehicle_type=metro&operatedBy=o-dqc-met

...and drop the resulting map of the DC Metro GeoJSON into **QGIS**
(<http://www.qgis.org/en/site/>). Hello Red Line!



Elsewhere

Can't find your favorite bus, train, gondola, or ferry in Transitland? **Please add it to the Transitland Feed Registry (<https://transit.land/news/2016/02/19/get-started-add-feeds.html>).**

Whether it's this weekend as part of a hackathon, next Monday as part of your "day job," or any other day that you're intrigued by open transit data, we welcome your involvement in Transitland!

· 04 March 2016 ·

**John Oram**

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

**Drew Dara-Abrams**

Drew leads Mapzen Mobility products (and aspires to being a flâneur).

© 2017 Mapzen

Helping make open mapping more inclusive

osm (/tag/osm)

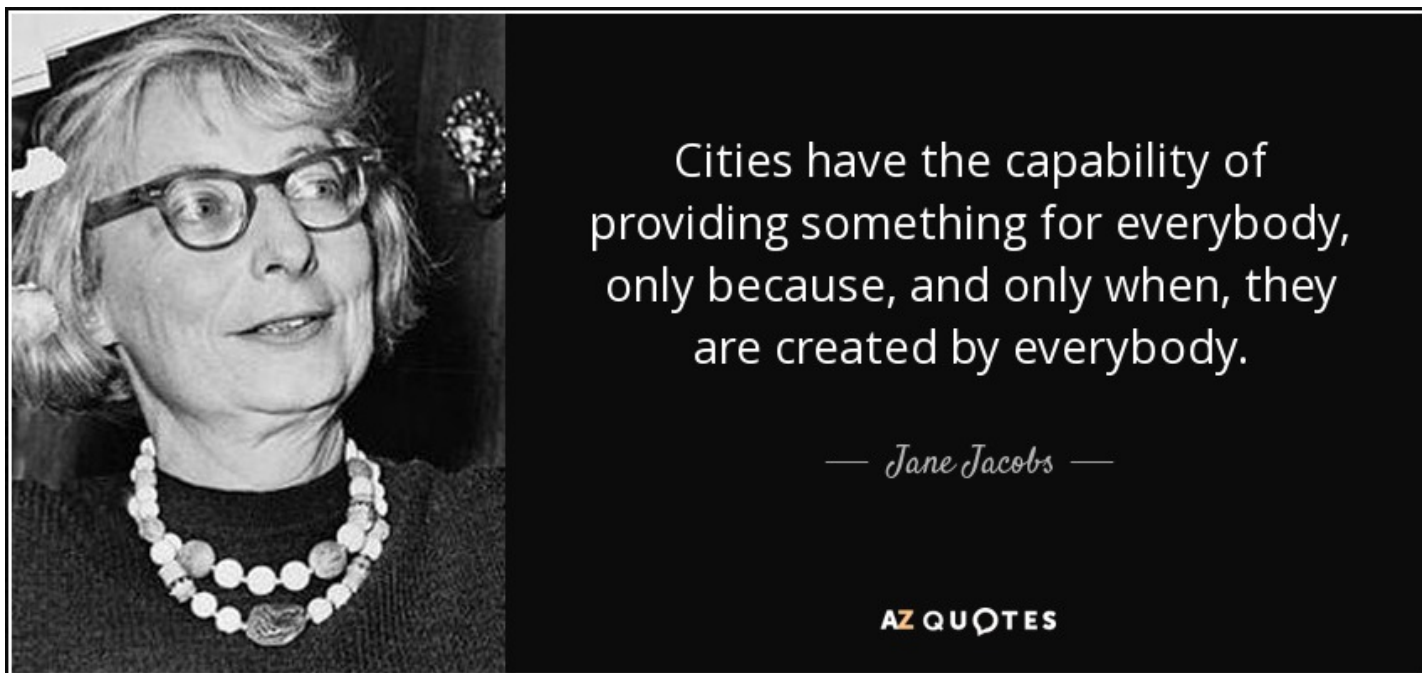


image courtesy of AZ-Quotes (<http://www.azquotes.com/quote/394402>)

OpenStreetMap is a lot of things to a lot of people, but some folks are getting left behind!

Diversity is sorely lacking among OpenStreetMap contributors. Many organizations and individuals are working to **shed light on this predicament** (<https://wiki.openstreetmap.org/wiki/Diversity>).

This **International Women's Day**, the OpenStreetMap community is setting out to increase the number of women and girls contributing. They're doing so through a **nation-wide Mapathon on Tuesday, March 8** (<https://openstreetmap.us/2016/02/womens-mapathon/>), which aims to introduce and encourage more female contributors to OSM. Organizations across the US are hosting mapping parties for women and girls who would like to contribute to OpenStreetMap.

“Because editing OSM gives participants a voice in how their communities are represented to the world, because digital literacy is a must-know skill, because women and girls around the world are disproportionately burdened with work that prevents their participation in “extra” activities, this International Women’s Day we ask every OSM community member around the world to contribute to gender parity.” – **Drishtie Patel, OSM US**
(<https://openstreetmap.us/2016/02/womens-mapathon/>)

We are excited to support this important cause, and to do so, Mapzen will be hosting a **Mapping Party** in our NYC office! We’d love to see more women and girls get introduced to OpenStreetMap and will be walking newcomers through their first edits. We are going to focus our editing efforts around inclusive **OSM tags** (http://wiki.openstreetmap.org/wiki/Tagging_in_Support_of_Women_and_Girls) that benefit a more diverse audience.

There will be healthy refreshments and snacks for all. Folks are welcome to stop by with the whole family. Younger kids can draw maps and the older ones can make their first OSM edit if they’re up for it!

So grab your friends and family, and head on over to our NYC office at **6:00pm** on **March 8th!** (see **event page** (<http://inclusivemaps.splashthat.com>) for details) We can’t wait to see **you!**

Please RSVP here (<http://inclusivemaps.splashthat.com>)!

· 07 March 2016 ·



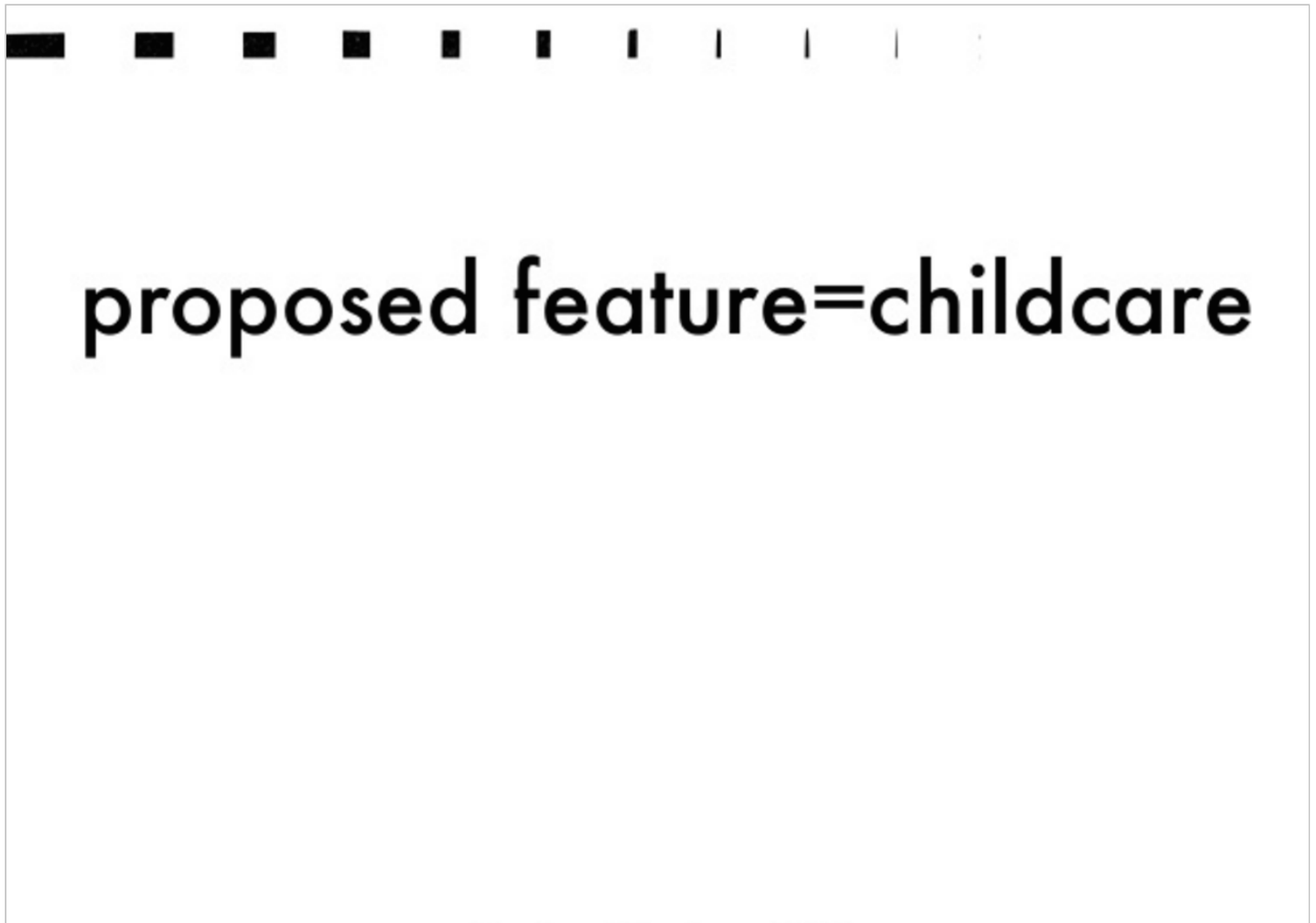
Diana Shkolnikov

Diana is the engineering director of Search@Mapzen and a proponent of mandatory fun, testing, education, and paleo, not necessarily in that order.

Our tiles are now more inclusive!

osm (/tag/osm)

A couple years ago I presented work by Monica Stephens about the Childcare tag in OpenStreetMap (<http://www.slideshare.net/apw217/changing-the-ratio>).



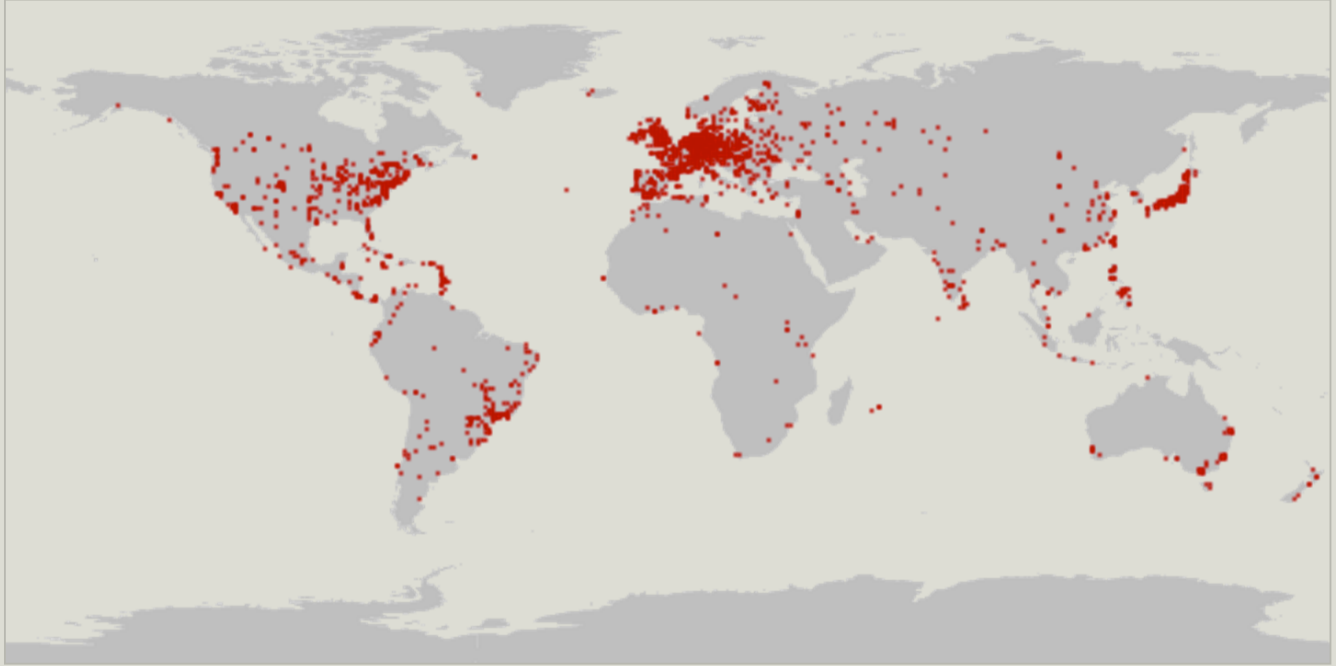
Update: we put kids **on the map** (<http://taginfo.openstreetmap.org/keys/amenity#map>)!

amenity=childcare

Describes a place where children of different ages are looked after

[Overview](#)[Combinations](#)[Map](#)[Wiki](#)[Projects](#)

Geographical distribution of this tag



OpenStreetMap started over 10 years ago in rebellion to who “owned the map” (aka the Ordnance Survey). The vision of OpenStreetMap was carried by a group of enthusiasts who mapped a world that was accessible and free to use. Their work was revolutionary and why we have an open map today and why we have open mapping communities the world over.

Like any creative work, a map reflects the views and needs of its authors. What might have first started in rebellion has multiplied in purpose. So as we consider all the many different ways OpenStreetMap now matters, we notice a number of features needed beyond the purposes of its original authorship.

In celebration of **International Women’s Day** (<http://www.internationalwomensday.com/>), cities around the world are holding **map events** (<http://www.internationalwomensday.com/Events>) to encourage the learning and the mapping for more inclusive demographics. The organizers have **identified a handful of tags** (https://wiki.openstreetmap.org/wiki/Tagging_in_Support_of_Women_and_Girls) that are not included in the oeuvre of official tags.

Today we announce the **inclusion of many of those tags in the Mapzen tile set** (<https://github.com/mapzen/vector-datasource/issues/526>).

Add new POIs for Women's Day #526



nvkelso opened this issue 25 days ago · 7 comments



nvkelso commented 25 days ago

Mapzen member

To support International Women's Day, let's add several new features (some of these are buildings, some are POIs, some might be on landuse polygons).

Mapzen vector tile change proposal:

- ✓ `amenity=social_facility` , where the kind is set to value of `social_facility=*` :
http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dsocial_facility - zoom 17, this needs to be tied in with a TileStache transform change.
- ✓ `amenity=clinic` : <http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dclinic> - zoom 17
- ✓ `amenity=doctors` : <http://wiki.openstreetmap.org/wiki/Tag:amenity%3Ddoctors> - zoom 17
- ✓ `amenity=kindergarten` : <http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dkindergarten> - zoom 17
- ✓ `amenity=childcare` : <http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dchildcare> - zoom 17
- ✓ `emergency=phone` : <http://wiki.openstreetmap.org/wiki/Tag:emergency%3Dphone> - zoom 18

/cc @awright

We encourage you to use these tags in your own depiction of the world and **add to them** (https://wiki.openstreetmap.org/wiki/Tagging_in_Support_of_Women_and_Girls). With you and these features, we hope OpenStreetMap can continue to be the most open map in the world and the most inclusive.

· 08 March 2016 ·



Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.

© 2017 Mapzen

New Year's Resolutions – Travel

targeted-editing (/tag/targeted-editing) **osm** (/tag/osm)

Maps are current as of Oct 2016.

Three months ago, when we were all ushering in 2016, did you resolve to travel more this year? You are not alone. “Travel more” New Year’s resolutions are very popular. With Spring Break upon us, help your fellow travel resolution enthusiasts by adding hotels and other types of lodging to OpenStreetMap.

There is probably a collection of hotels in your local area that you are very familiar with, and there may be a few additional ones that you would like to know more about. This is your chance to do some investigating which could lead to your next staycation or an easier time finding places to put up friends and family when they all decide to visit at once. No doubt there are places you have stayed in the past that left a lasting impression (for better or for worse). Now is your chance to share your local knowledge from around town, or towns you have visited over the years.

Our last few posts in this series are showing us how business listings and other points of interest are slowly gaining momentum in OpenStreetMap. Curious to know how prevalent hotels, motels, hostels, and other forms of lodging are in OpenStreetmap? Check out this table which includes our New Year’s Resolutions cities, but this time we are including phone numbers instead of hours since many lodging options recieve guests at any hour. Go ahead and click the headers to sort the table!

	Lodging Features	with Addresses	with Website	with Phone Number
Auckland	131	7%	2%	1%
Dublin	321	36%	38%	21%
Hong Kong	752	39%	10%	9%
Melbourne	254	39%	54%	24%

	Lodging Features	with Addresses	with Website	with Phone Number
Mexico City	172	17%	11%	13%
Mumbai	251	4%	4%	6%
San Francisco	342	53%	33%	27%
São Paulo	270	44%	24%	27%
Seoul	955	5%	4%	4%
Singapore	428	38%	24%	20%
Stockholm	485	12%	46%	27%
Vancouver	92	63%	22%	17%

*SQL queries (https://github.com/mapzen-data/targeted-editing/blob/gh-pages/queries/lodging_sql.sql) for those interested in generating stats for additional cities.

Wow! So many lodging options in Seoul, but so few associated attributes. This is an amazing opportunity for editors. For websites and phone numbers, Melbourne and Stockholm are leading the pack, with Melbourne edging out Stockholm just a bit with it's additional address details. A street address on a feature adds so much value for many types of map services.

Helpful Tags

If you are behind in your vacation planning, get inspired by adding or enhancing lodging related businesses in OpenStreetMap! Need some help choosing tags? Here are some suggestions which link to their corresponding OpenStreetMap wiki pages:

- **name=*** (<https://wiki.openstreetmap.org/wiki/Key:name>)
- **addr:housenumber=*** (<https://wiki.openstreetmap.org/wiki/Key:addr>)
- Polygons to represent the entire grounds:
 - **tourism=hotel** (<https://wiki.openstreetmap.org/wiki/Tag:tourism%3Dhotel>)
traditionally full service hotels.
 - **tourism=motel** (<https://wiki.openstreetmap.org/wiki/Tag:tourism%3Dmotel>)
traditionally smaller hotels with parking in front of each room.
 - **tourism=hostel** (<https://wiki.openstreetmap.org/wiki/Tag:tourism%3Dhostel>)
traditionally lower cost lodging with shared communal spaces.
 - **tourism=chalet** (<https://wiki.openstreetmap.org/wiki/Tag:tourism%3Dchalet>)
small cottages with basic accommodations.

- Points or polygons to represent the buildings
 - **building=*** (<https://wiki.openstreetmap.org/wiki/Buildings>)
 - examples: building=hotel, building=motel, building=hostel, etc
- **website=*** (<https://wiki.openstreetmap.org/wiki/Key:website>)
- **phone=*** (<https://wiki.openstreetmap.org/wiki/Key:phone>)

Where to Start

Do you travel for business and stay at the same hotels regularly? Enhance those first. Do you have go-to hotels you suggest when people are visiting from out of town? Save those details in OpenStreetMap for future retrieval. Maybe you had a particularly memorable time at the places you booked for your last vacation. Are they represented in OpenStreetMap?

Interactive Map

Would it be helpful to see where the current lodging related features are in OpenStreetMap? There's a map for that! Hover over any of the bright blue highlighted features to bring up an info bubble with links to editing tools that can be used to add additional tags. (While you pan and zoom to your neighborhood, note that larger lodging features show up at zoom level 15, and many more show up at zoom 16.)



Leaflet | Geocoding by Mapzen

This map is interactive! Open full screen ↗ (<https://mapzen-data.github.io/targeted-editing/te-lodging/map/index.html?hotel>)

If you're not familiar with Vancouver, search for your city! Is your go-to hotel missing all together? Shift-click anywhere on the map to open iD or open your favorite OpenStreetMap editor to start mapping!

If you are trying to decide between creating a point or a polygon to represent a feature, a point is great when the lodging is in a building that contains other businesses and/or residences. If your lodging occupies the entire building, digitize a building polygon if it doesn't already exist and add your tags to it.

Look for more posts in the **New Year's Resolution** (<https://mapzen.com/tag/new-years-resolutions>) series soon!

Getting Started with OpenStreetMap

Need instructions on how to edit with iD? Here are some links to outstanding tutorials from LearnOSM, the OpenStreetMap wiki, and the United States Department of State's Humanitarian Information Unit:

- **LearnOSM** (<http://learnosm.org/en/>)
- **OSM Beginners' Guide** (http://wiki.openstreetmap.org/wiki/Beginners'_guide)
- **MapGive Learn to Map** (<http://mapgive.state.gov/learn-to-map/>)

Are you a mapping wiz and interested in a more advanced editor? Try out **JOSM** (<https://josm.openstreetmap.de>) with **excellent documentation** (<https://www.mapbox.com/blog/osm-mapping-guide/>) from Mapbox.

Editing with a Mobile Device

With points of interest, I bet you are interested in a mobile app to edit OpenStreetMap while you are on the go.

iOS users can check out **Go Map!!** (<https://appsto.re/us/dafwj.i>) by Bryce Cogswell. This free editor allows you to add features and edit existing features. Check out the **Go Map!!** (http://wiki.openstreetmap.org/wiki/Go_Map!!) wiki page for more details. **Pushpin** (<http://www.pushpinosm.org>) by Fulcrum is another excellent iOS editor for OpenStreetMap points of interest.

Looking for an Android equivalent? **Vespucci** (<https://play.google.com/store/apps/details?id=de.blau.android>) by Marcus Wolschon is a great option. Check out the **Vespucci** (<http://vespucci.io>) home page for documentation and tutorials.

Get outside, increase your local knowledge, and share it with the world through your OpenStreetMap edits!

*Check out all the posts in the **Targeted Editing series** (<https://mapzen.com/tag/targeted-editing>):*

- **Airports** (<https://mapzen.com/blog/targeted-editing-airport-polygons>)
- **Banks** (<https://mapzen.com/blog/new-years-resolutions-money/>)
- **Building Names** (<https://mapzen.com/blog/targeted-editing-name-that-building>)
- **Campus Mapping** (<https://mapzen.com/blog/targeted-editing-campus-mapping/>)
- **Cycleways** (<https://mapzen.com/blog/targeted-editing-cycleways/>)
- **Fitness** (<https://mapzen.com/blog/new-years-resolutions-fitness>)
- **Groceries** (<https://mapzen.com/blog/new-years-resolutions-groceries>)
- **Hospitals** (<https://mapzen.com/blog/targeted-editing-hospital-polygons>)

- **Schools** (<https://mapzen.com/blog/targeted-editing-school-polygons>)
- **Stadiums & Parking** (<https://mapzen.com/blog/targeted-editing-tailgate-mania>)
- **Street Names** (<https://mapzen.com/blog/targeted-editing-no-name-roads>)
- **Transit Colours** (<https://mapzen.com/blog/targeted-editing-transit-colours>)
- **Travel & Lodging** (<https://mapzen.com/blog/new-years-resolutions-travel>)

· 10 March 2016 ·



Indy Hurt

Indy was our resident data scientist lending her geographic expertise to all things "open".

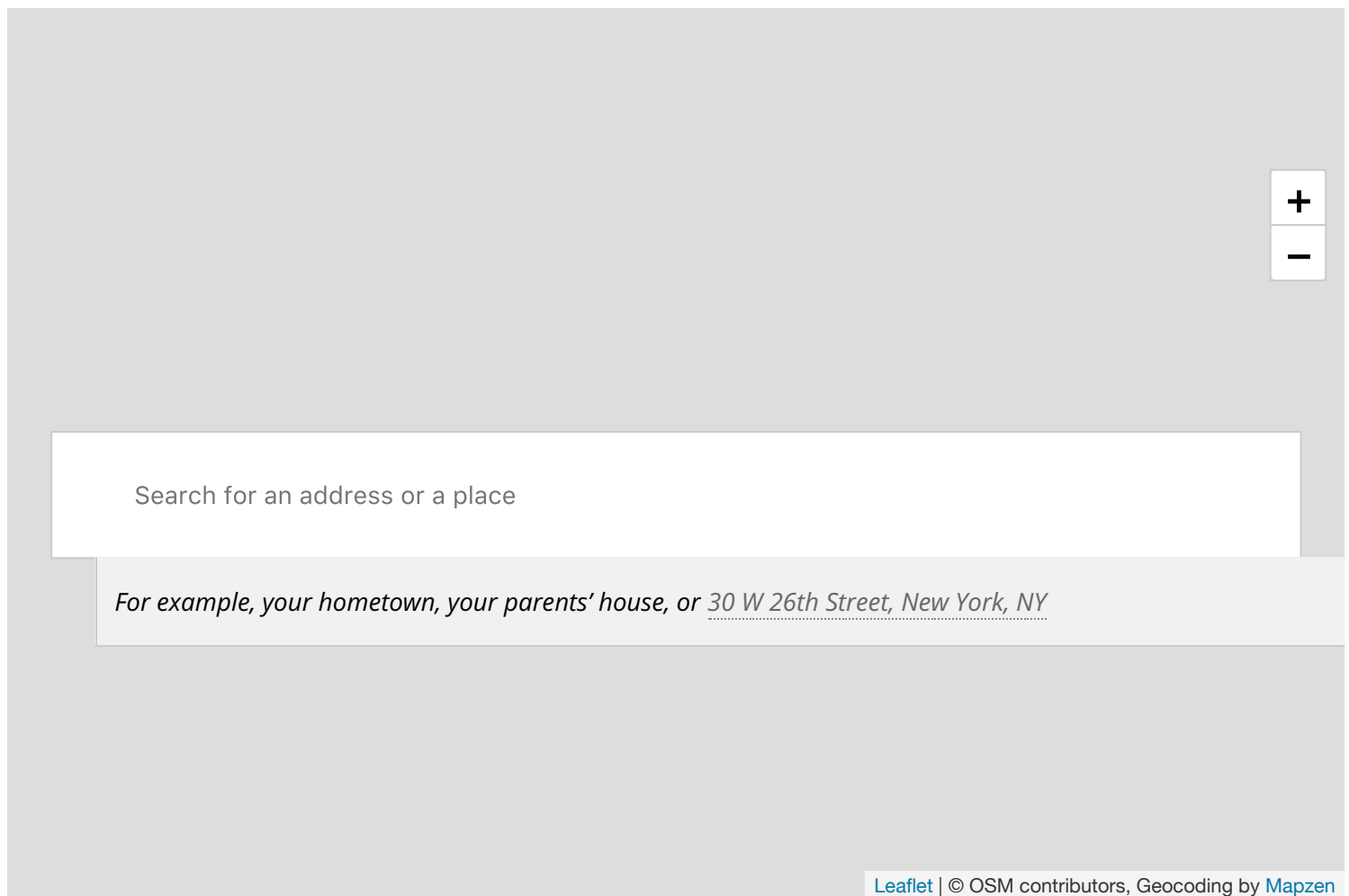
© 2017 Mapzen

Building Mapzen Search's product demo

search (/tag/search) demo (/tag/demo)

Hello Mapzen newsletter readers! An old URL snuck in – here is that Guardian article (<https://www.theguardian.com/cities/2016/sep/02/this-interactive-map-lets-you-watch-who-commutes-to-your-city-every-day>). Sorry about that! (But this post is pretty good too.)

In the five months since we **launched** (<https://mapzen.com/blog/this-is-it-mapzen-search-is-now-live>) the production service of Mapzen Search, we've talked to many of our users and learned quite a lot about how people use it. So, with new feedback in mind, it was time to update our website with a brand-new demo that shows off what it can do. You can take a look for yourself on the **Mapzen Search project page** (<https://mapzen.com/projects/search/>), or better yet, just keep your eyeballs on this blog post and I'll go ahead and embed that for you.



*This looks better on the project page. **Go there now!** ↗ (<https://mapzen.com/projects/search>)*

We had two primary goals in mind:

1. **Keep it simple.** It should tease the service, not be a full-fledged search application.
2. **Show off the API.** It should demonstrate which service endpoints are active and what data they return.

It turns out that finding that proper balance between these two goals is much harder than you'd think once combined into one project. With the Mapzen Search **Leaflet geocoder plugin** (<https://github.com/mapzen/leaflet-geocoder>) in our toolkit, we could easily build a prototype that hits both goals in broad strokes in just a couple of days. But finessing the interface took about four times as long (as Joel Spolsky says of craftsmanship, **"fixing a 1% defect takes 500% effort"** (<http://www.joelonsoftware.com/articles/Craftsmanship.html>)) —and if you look *too* closely, you still might find the edge cases we haven't worked out yet).

The self-guided tour

Early on, we decided to resist the urge to create a “guided tour” experience, which would have the effect of preventing visitors from making esoteric search queries that might not return correct results from existing open data sources. The reality is that the data is in the best shape it's ever been, and we're constantly improving it. It's important to show that evolution as time goes on, so the search box allows you to explore whatever location you'd like.

But we didn't want people to sit around thinking of something to search for, so we put a prompt underneath the search box. Many people already type in their home address or the town they grew up in, because it's a familiar setting and easy to gauge against the accuracy of the geocoder. But in case you're struggling to think of an address, we've provided an example you can use (it's the address of Mapzen headquarters in New York City). Click on it, and it simulates typing for you.

Autocomplete vs. search

As you type (or, if you prefer robots to do all your work, *as JavaScript simulates your keystrokes*), we send each new input to the **/autocomplete endpoint** (<https://mapzen.com/documentation/search/autocomplete/>) of Mapzen Search, providing real-time, predictive feedback about the query you're trying to make. At any point, you can press the Enter or Return key to send **a query to the /search endpoint** (<https://mapzen.com/documentation/search/search/>), for the set of results that match your query.

Confused about the difference? Don't worry, you're not the only one! One way to illustrate the difference between the two endpoints is this: imagine that you're typing the query “yolo.” When you're sending that query to **/autocomplete**, you might not be done typing yet, so the first four results from Mapzen Search might be:

- Yolöten, Mary, Turkmenistan
- Yolombó, Antioquia, Colombia
- Yolo County, CA
- Yolonan, Chamula, Mexico

But maybe you are trying to search for “yolo”, as in Yolo County, California. So results from **/search** would include:

- Yolo, Yolo County, CA
- Yolo County, CA
- La Puerta del Yolo, Cuautepéc de Hinojosa, Mexico
- Yolo, Mopti, Mali

Here's another way to think about it: suppose you're typing, and so far you have typed "Lond". Chances are, you really mean London, England, and it's less likely that you meant Lond, Pakistan, so `/autocomplete` predicts the most likely anticipated results and surfaces those for you. (And if you really did want Lond, Pakistan, `/search` has your back.)

Nuts and bolts

We wanted to make it really clear what's actually happening behind the scenes when our demo makes a request and shows the raw data that gets sent back. So, for every request, we show you exactly the API endpoint our demo is querying, for instance:

```
https://search.mapzen.com/v1/search?text=San%20Diego
```

(We actually mask the API key, which is otherwise required for every request.)

But, because we're just showing a basic use, we aren't making any queries that include the adjustments that you might normally want in a full-fledged application like **narrowing search results to your country** (<https://mapzen.com/documentation/search/search/#combine-boundary-search-and-prioritization>) or **filtering by data type** (<https://mapzen.com/documentation/search/search/#filter-your-search>). Still, the results you get back will always be in the same format. Glancing through the returned JSON object, you can very quickly understand how you might craft your application to use the data that we serve.

Using the Leaflet geocoder plugin

As I've mentioned earlier, using our own homegrown **Leaflet geocoder plugin** (<https://github.com/mapzen/leaflet-geocoder>) allowed the team to get up and running with a prototype demo very quickly. In the process, we were even able to squash a few bugs and include support for the `/place` endpoint (<https://mapzen.com/documentation/search/place/>), which led to our most **recent release of the plugin** (<https://github.com/mapzen/leaflet-geocoder/releases/tag/v1.5.1>).

But we've certainly pushed the limits of what it could do for us. There's a lot of custom UI that's required, and the plugin was never designed to support this amount of functionality to be bolted to it. For instance, the results list that appears after you type in some search term is attached to the boxes where we output the JSON data and API query endpoint. The plugin creates all of its required elements inside of Leaflet, so how did we get it to display anything else?

To do this, we use the plugin's **event listeners** (<https://github.com/mapzen/leaflet-geocoder#events>) to respond to certain actions, like whenever results are returned. Each event provides access to the raw data and query information, so that's how we can reconstruct that information in a separate part of the demo.

But that separate area doesn't otherwise know anything about Leaflet, and Leaflet doesn't know where the results are supposed to go. To gather these two separate parts in one place, we do a little manipulation of the HTML using JavaScript:

```
var resultsEl = document.querySelector('.leaflet-pelias-results');  
var resultsContainerEl = document.getElementById('results-placeholder');  
resultsContainerEl.parentNode.replaceChild(resultsEl, resultsContainerEl);
```

Basically, we move the results list outside of Leaflet and into another part of the page!

Since the geocoder plugin “remembers” the HTML element, it doesn't matter where on the page it actually exists. We can move it anywhere, and the plugin still knows how to get to it! You can actually do this with the geocoder input itself, which is something we didn't do for this demo, but it would have worked just as well.

Ultimately, though, this experience has taught us that when it comes to slightly more complex work, we spent more time than we needed to (and nursed several headaches) trying to override assumptions we made in the geocoder plugin. We designed the geocoder UI to allow it to work smoothly right out of the box for most users, across as many different devices and browsers as possible. But for this demo, the UI surrounding the geocoder is very different, so it felt very much like taking a very nicely made square peg but trying to force it into a tangentially related round hole. In future iterations of the demo (where we fix bugs, clean up code, and add new search features as they evolve), we could fork the plugin code rather than depend on the plugin as a dependency. This will make the demo much easier to maintain, easier to modify, and not make as many tradeoffs in usability for underlying technical reasons.

Build, measure, learn

The year of 2015 was a year of releasing a lot of product into the wild. The year of 2016 will be a year of learning from 2015 and iterating so that we can make our products even better. You'll definitely see this from the Mapzen Search team and also in the demo we built to show it off, so stay tuned!

Note: This post was updated on May 31, 2017 to remove outdated API request links.

· 11 March 2016 ·



Lou Huang

Lou is a protagonist of an open world.

© 2017 Mapzen

Eraser Map – join the beta

mobile (/tag/mobile) **erasermap** (/tag/erasermap) **privacy** (/tag/privacy)

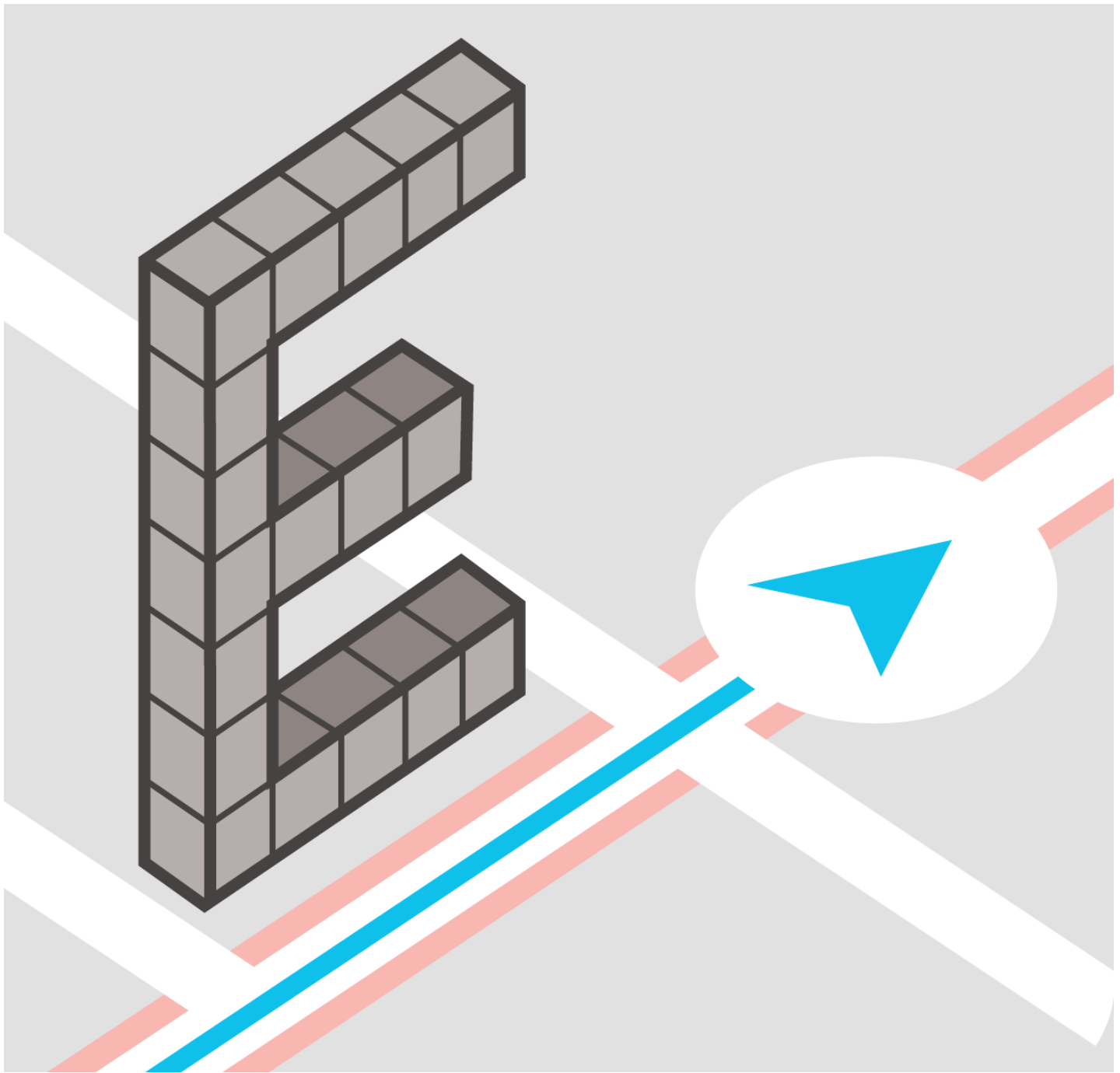
android (/tag/android)

You can't test a map sitting at your desk. You really have to take it with you outside. Last year, we began building a set of SDKs for our services and packaging them into a reference application on Android. This allowed us to do things like take the **Mapzen Turn-by-Turn** (<https://mapzen.com/projects/valhalla>) service out for an afternoon drive or to see how well **Mapzen Search** (<https://mapzen.com/projects/search?lng=-73.99223&lat=40.75545&zoom=12>) can find that **weird address in Queens** (<https://mapzen.com/blog/meaningful-geocoding-address-search-the-two-core-principles-of-geocoding>) while you're standing a few blocks away.

As we tested our app, it became clear that the quality of the open data used to power it was improving rapidly. As the months passed, every time we would open and use this app we could *feel* the address data in search, the road network in navigation, and the details of the map improving. The work being done by the open data community is amazing.

While we believe in open data, some data should not be collected at all. A company we really admire is **DuckDuckGo** (<https://duckduckgo.com/>), a search engine that focuses on user privacy. As we saw open data quality improve, we wondered if we could use our SDKs to build a mobile mapping application that could make the same promise as DuckDuckGo: not to log or track our users, at a time when user privacy is a major concern. Since Mapzen runs its own map stack and is not dependent on third-party mapping providers, we can follow through on promises we make about how we handle user data. We even went so far as to create **LOST** (<https://mapzen.com/blog/lets-get-lost>), a drop-in replacement for Google Play Location Services, so your location is kept as private as possible.

We saw an opportunity to improve privacy in mobile mapping, and with that in mind we'd like to invite you to help us beta test **Eraser Map** (<http://www.erasermap.com>).



Eraser Map is a privacy-focused mapping application that tracks nothing about you and has no user accounts. The entirety of Eraser Map is built using open-source software and open data. This means users and developers can inspect the app, improve it, and call out any issues they find. We believe that providing all our work as free and open software increases scrutiny and trust in our products, especially those focused on user privacy.

So what goes into a mapping application? A whole lot. Take a look at our work:

- Mobile app: check out **Eraser Map code on Github** (<https://github.com/mapzen/eraser-map>)

- Map display: we use **Mapzen Vector Tiles** (<https://mapzen.com/projects/vector-tiles>) with the lovely **Tangram-ES** (<https://github.com/tangrams/tangram-es>) rendering engine
- Search engine: we use **Mapzen Search** (<https://mapzen.com/projects/search/?lng=-73.99223&lat=40.75545&zoom=12>) built on **Pelias** (<https://github.com/pelias>)
- Turn-by-turn navigation: we use **Mapzen Turn-by-Turn** (<https://mapzen.com/projects/valhalla>) powered by **Valhalla** (<https://github.com/valhalla>)
- Map data: **OpenStreetMap** (<http://www.openstreetmap.org/>) contributors provide an incredible global map dataset, which we augment with **OpenAddresses** (<https://openaddresses.io/>) and **Who's on First** (<https://whosonfirst.mapzen.com/>), our open global gazetteer

We'd love for you to use any of these in your own projects, contribute to them, or just **get in touch** (<mailto:hello@mapzen.com>).

We have taken this app from Korea, to Argentina, to India and all over the US (and even Canada) and we'd love you take it even more places. Sign up to become an Android beta tester over on **erasermap.com** (<http://www.erasermap.com>).

PS: The irony is not lost on us that we're asking for your email addresses to join the beta program but that's a requirement of the Google Play beta program. We'll allow sideloading once we launch.

· 14 March 2016 ·



Mike Cunningham

I'm Street View famous in two cities and I do product @mapzen.



Chuck Greb

Chuck is an open source enthusiast, test-driven evangelist, and clean code connoisseur of all things mobile.

© 2017 Mapzen

Targeted Editing – Transit Colours

osm (/tag/osm) **targeted-editing** (/tag/targeted-editing)

Maps are current as of Oct 2016. To view newer transit data **go to Refill** (<http://tangrams.github.io/refill-style/#12/40.7381/-73.9541>), “show” transit in the widget in the top right corner, and then tick the “interactive” checkbox to introspect transit lines.

Transit colours help distinguish lines!


Welcome to our **Targeted Editing** series where today you can hop on your favorite transit line, or make **OpenStreetMap** (<http://www.openstreetmap.org>) data divine! (Both take about the same amount of time.)

Read on for some basic uses for transit line colours, why I am using the British spelling, and where you can check to see if you can use your local knowledge to add colours where they are needed.

How can transit line colours be used?

1. **Help with map rendering.** Transit line colours can be used to make custom maps designed specifically for transit users.
2. **Help with wayfinding.** A line colour on a map can help a traveler quickly understand generally where a route goes, without needing prior spatial knowledge of the origin or destination stops.

Are there transit features in OpenStreetMap? Yes!

But how do we quantify them? Let me count the **ways** (<http://wiki.openstreetmap.org/wiki/Way>)...  actually, no, let's not do that. That would only tell us how many individual segments we have, which is a bit abstract. It effectively translates to the number of segments editors have digitized. Not trip segments, mind you, just individual ways with unique ids. Instead, let's look at kilometers.

Kilometers of Transit Lines

	Railway Relations	Railway Ways	Route Relations	Route Ways
Beijing		4,835	11,655	
Berlin		10,385	45,289	
Boston		663	994	
DC	25	3,763	16,018	
London	106	12,738	50,489	2
Moscow		11,070	69,602	
New York		6,025	21,745	
Paris	3	9,013	44,841	0.10
San Francisco	21	1,381	6,372	
São Paulo		1,909	8,642	
Sydney		1,576	6,116	0.58
Tokyo	69	11,893	37,831	0.14

***SQL queries (https://github.com/mapzen-data/targeted-editing/blob/gh-pages/queries/transit_sql.sql)** for those interested in generating stats for additional cities.

What's happening here?

Transit features are relatively well represented in our sample of cities. The railway key is more often associated with ways, whereas the route key is more often associated with relations.

What does it mean for one city to have more kilometers than another city? Diversity of added transit lines could be one factor. Separate lines for inbound and outbound routes can be another. Yet a third factor could be the detail of editing because a very straight line between point A and point B will be shorter than a curvy line between the same point A and point B. Last but not least, these cities are different in size with varying public transportation networks. Given the area of each of these cities (as determined by the Mapzen metro extract, not reality) these numbers are all impressive.

What does it mean when the length of relations far exceeds the length of ways? The most dominant explanation goes to **Osm2pgsql** (<http://wiki.openstreetmap.org/wiki/Osm2pgsql>) and the consistency of tags editors apply. The data for this post was imported into a PostgreSQL database using Osm2pgsql. When a relation and it's members have the same tags, Osm2pgsql drops the members during the import process.

The table above is a result of looking at the collective distance across many transit types (buses, funiculars, light rails, monorails, rails, railways, subways, trains, trams, and trolleybuses).

Breaking this down further, at least some colour is applied in every city, and you are more likely to find it applied to **route relations**. This makes sense because when a relation is present, it is much easier to add additional tags to it. Tags on a relation apply to all the ways that are members of the relation. With this in mind let's review route relations, and the transit types that are more likely to be referred to with colours. Think of things like **"The Red Line"** (<http://www.transitchicago.com/redline/>) in Chicago.

Percent of Route Relations with Colour

	Light Rail	Subway	Train	Tram
Beijing		72%		
Berlin	100%	100%	61%	81%
Boston		100%	81%	13%
DC		88%	83%	
London		100%	51%	100%
Moscow			2%	
New York	72%	98%	94%	
Paris	100%	96%	94%	96%
San Francisco	8%	100%	57%	19%
São Paulo		100%	100%	
Sydney	100%		48%	
Tokyo	19%	92%	12%	

What's happening here?

Spatial variability, that's what!

Figuring this out is like a scavenger hunt... only indoors... with a database. Ok, so not like a scavenger hunt, but definitely exploratory data analysis. Are we missing some things? Probably. The queries are specifically looking for relations with the `route` key. In Moscow, some of the colour is only applied to the `route_master` which works well when relations representing the inbound and outbound routes are members.

It is easy to focus on relations that are tagged `route=light_rail` , `route=subway` , `route=train` , or `route=tram` , but each city can be different. For example, Tokyo also has 133 kilometers of relations that are tagged `railway=subway` and half of them have a populated colour tag. As you can see from the table above, Tokyo also has relations that are tagged `route=subway` . In other cases, a relation doesn't exist at all, but an editor has taken the time to add colour to the individual ways.

Helpful Tags

- **name** (<http://wiki.openstreetmap.org/wiki/Names>)=*
- **ref** (<http://wiki.openstreetmap.org/wiki/Key:ref>)=*
- **colour** (<http://wiki.openstreetmap.org/wiki/Key:colour>)=*
 - Tips (because consistency is key)
 - Use an RGB colour code (hex triplet) preceded by a hash or pound symbol with lower case letters where present.
 - YES: #ff0000
 - NO: FF0000
 - Alternatively, use a CSS color name in all lower case letters. No spaces, no underscores.
 - YES: mediumblue
 - NO: medium blue
 - NO: Medium_Blue

Again, the best place to add the colour tag is to the **route relation**. Be sure to add it to inbound and outbound route relations where present. If a `route_master` is present, add the colour tag to it, too. A `route_master` is used to group relations. An example in transit would be a `route_master` that groups the inbound and outbound route relations. If no relation is present, this is your opportunity to add one. Adding the colour tag to the individual ways should be a last resort.

How do you find the route relation? Using iD, you can always click any way, and if it is a member of a relation, you will find a link to the relation listed below the tags when you are in edit mode. Check the left side of the screen, and scroll all the way down to the bottom. Sometimes you will find multiple relations.

For details on relations in general and the specifics of `route` and `route_master` relations, explore these wiki links:

- **relation** (<http://wiki.openstreetmap.org/wiki/Relation>)
- **route** (<http://wiki.openstreetmap.org/wiki/Relation:route>)=*
- **route_master** (http://wiki.openstreetmap.org/wiki/Relation:route_master)=*

A promise to explain why I am using the **British spelling** (https://en.wikipedia.org/wiki/American_and_British_English_spelling_differences) for the word “colour” begins now:

In the simplest terms, the British spelling is the preferred spelling for keys and values (tags, collectively) in OpenStreetMap. Nearly all* of the data explored for this post had the British spelling when colour was present and evaluated. Other British spellings to watch out for in OpenStreetMap include centre, jewellery, kerb, and neighbourhood.

***See the queries** (https://github.com/mapzen-data/targeted-editing/blob/gh-pages/queries/transit_sql.sql) for additional tags that have been used to represent colour.

Where to Start

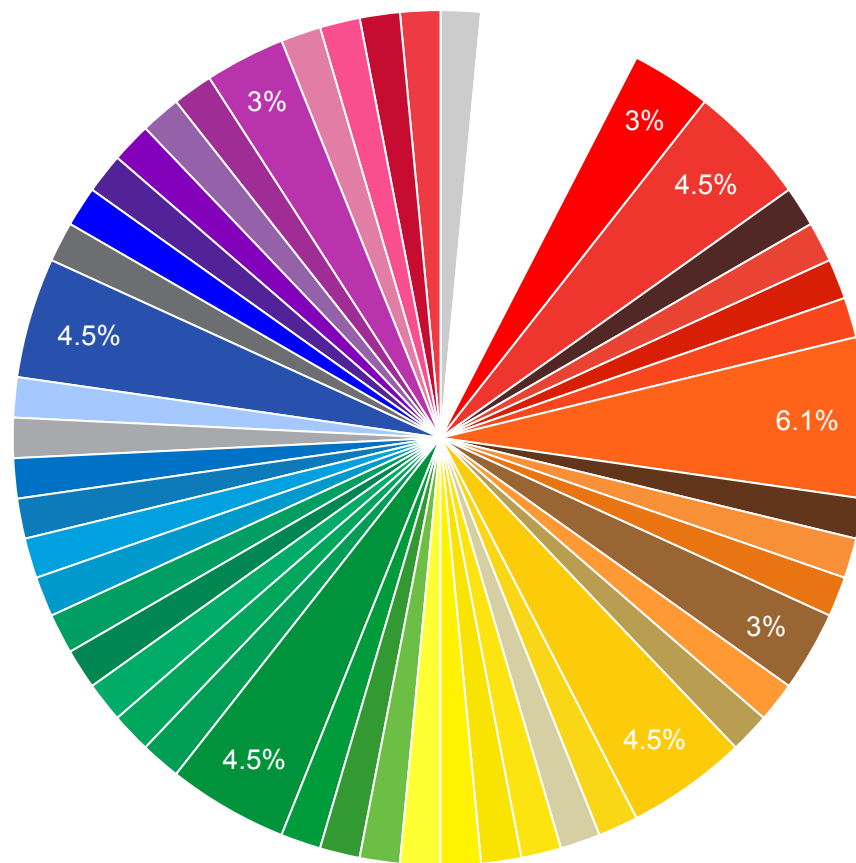
The best place to start is always where you have local knowledge. If you are familiar with a set of transit routes that are commonly referred to by colour and it would make sense to symbolize the routes with these well known colours, please review them in OpenStreetMap to ensure that they have been tagged properly.

It never hurts to augment your local knowledge with a bit of research for verification. Make notes when you are taking public transportation, and seek out authoritative sources. Mapzen’s **Transitland Feed Registry** (<https://transit.land/feed-registry/>) is another excellent resource.

The Feed Registry contains a steadily growing collection of **GTFS** (https://en.wikipedia.org/wiki/General_Transit_Feed_Specification) data, and some of the feeds include route colour! At the time of this writing, our Feed Registry had 159 GTFS feeds for metro lines with populated route colour data, and this is just a subset of the total number of

Metro Line Route Colours

from Transitland's Feed Registry



● #cccccc #ffffff ● #ff0000 ● #ee352e ◀ 1/16 ▶

Special thanks to Transitland engineers **Alex** (<https://github.com/doublestranded>) & **Ian** (<https://github.com/irees>) for querying the Transitland Feed Registry API and sorting the colors for this post! If you are interested in the magic of this sorting, check out Ian's **transitland_route_colors** gist (<https://gist.github.com/irees/f70babd3f6f9580d67bc>). We all had a lot of fun querying the Transitland API. Go ahead, give it a try! Here's a sample URL to get you started (**click to download the 4.6MB JSON file** (http://transit.land/api/v1/routes.json?vehicle_type=metro&per_page=1000)):

```
http://transit.land/api/v1/routes.json?vehicle_type=metro&per_page=1000
```

If “query the API” sounds like Greek to you, the user interface can be used to visualize data from the Transitland Feed Registry. When you land on the **Transitland Feed Registry** (<https://transit.land/feed-registry/>) page, follow the link in the Operator Name column to the agency you are interested in, click the “view” link for the routes if there is one, and wait a few moments for the data to render in the geojson.io viewer. Choose the table view and look for the title and stroke columns.

You can also download the GeoJSON directly via the API and open it in your map renderer of choice (**click to download the 4.6 MB GeoJSON file** (http://transit.land/api/v1/routes.geojson?vehicle_type=metro&per_page=1000)):

```
http://transit.land/api/v1/routes.geojson?vehicle_type=metro&per_page=1000
```

Here is what you will see for the Transit Authority of River City in Louisville, Kentucky:

The screenshot shows two browser windows. The top window is the Transitland website, displaying the 'Feed Registry' for the 'Transit Authority of River City (TARC)'. It lists various data types like Routes and Stops, each with a 'JSON, GeoJSON (view)' link. A black arrow points from the 'Routes' link to the bottom window. The bottom window is the 'geojson.io' interface, showing a map of Louisville, KY. A black arrow points from the 'Table' tab in the top right of the geojson.io interface to the URL box above it.

<https://transit.land/api/v1/routes.geojson?operatedBy=o-dng1-transitauthorityofrivercity>

You can **click to download the 2.4 MB GeoJSON file** (<https://transit.land/api/v1/routes.geojson?operatedBy=o-dng1-transitauthorityofrivercity>) or open in **geojson.io** (<http://geojson.io/#data=data:text/x-url,http%3A%2F%2Ftransit.land%2Fapi%2Fv1%2Froutes.geojson%3FoperatedBy%3Do-dng1-transitauthorityofrivercity>) (be patient, it will take a few seconds to render).

If you find a feed that you can use to verify your local knowledge, be sure to check the license to make sure it is permissible to add the data to OpenStreetMap. Most importantly, include reference to the agency with the **source** (<http://wiki.openstreetmap.org/wiki/Key:source>) key.

Interactive Map

Would it be helpful to see where colour is potentially missing from your favorite transit line in OpenStreetMap? There's a map for that! Hover over any of the bright blue highlighted features to bring up an info bubble with links to editing tools that can be used to add additional tags.



[Leaflet](#) | Geocoding by [Mapzen](#)

This map is interactive! Open full screen ↗ (<https://mapzen-data.github.io/targeted-editing/te-transit-colours/map/?transit>)

If you are not familiar with Washington DC, open the full screen map and search for your city! You might even find your favorite route missing all together. Sometimes you will notice a collection of station icons without a route connecting them. Of course, if you see anything missing, you can always shift+click the map to launch the iD editor.

Getting Started with OpenStreetMap

Need instructions on how to edit with iD? Here are some links to outstanding tutorials from LearnOSM, the OpenStreetMap wiki, and the United States Department of State's Humanitarian Information Unit:

- **LearnOSM** (<http://learnosm.org/en/>)

- **OSM Beginners' Guide** (http://wiki.openstreetmap.org/wiki/Beginners'_guide)
- **MapGive Learn to Map** (<http://mapgive.state.gov/learn-to-map/>)

Are you a mapping wiz and interested in a more advanced editor? Try out **JOSM** (<https://josm.openstreetmap.de>) with **excellent documentation** (<https://www.mapbox.com/blog/osm-mapping-guide/>) from Mapbox.

Puzzled by the mystery of relations? InaSAFE has a **JOSM tutorial featuring relations** (<http://docs.inasafe.org/en/training/old-training/intermediate/osm/301-advanced-editing.html>).

Get outside, increase your local knowledge, and share it with the world through your OpenStreetMap edits!

Check out all the posts in the **Targeted Editing series** (<https://mapzen.com/tag/targeted-editing>):

- **Airports** (<https://mapzen.com/blog/targeted-editing-airport-polygons>)
- **Banks** (<https://mapzen.com/blog/new-years-resolutions-money/>)
- **Building Names** (<https://mapzen.com/blog/targeted-editing-name-that-building>)
- **Campus Mapping** (<https://mapzen.com/blog/targeted-editing-campus-mapping/>)
- **Cycleways** (<https://mapzen.com/blog/targeted-editing-cycleways/>)
- **Fitness** (<https://mapzen.com/blog/new-years-resolutions-fitness>)
- **Groceries** (<https://mapzen.com/blog/new-years-resolutions-groceries>)
- **Hospitals** (<https://mapzen.com/blog/targeted-editing-hospital-polygons>)
- **Schools** (<https://mapzen.com/blog/targeted-editing-school-polygons>)
- **Stadiums & Parking** (<https://mapzen.com/blog/targeted-editing-tailgate-mania>)
- **Street Names** (<https://mapzen.com/blog/targeted-editing-no-name-roads>)
- **Transit Colours** (<https://mapzen.com/blog/targeted-editing-transit-colours>)
- **Travel & Lodging** (<https://mapzen.com/blog/new-years-resolutions-travel>)

· 18 March 2016 ·



Indy Hurt

Indy was our resident data scientist lending her geographic expertise to all things "open".

© 2017 Mapzen

Mapping Mountains

tangram ([/tag/tangram](#)) **data** ([/tag/data](#)) **demo** ([/tag/demo](#)) **terrain** ([/tag/terrain](#))

I've been spending a lot of time over the mountains of Northern California lately. To view mountains from above is to journey through *time itself*: over ancient shorelines, the trails of glaciers, the marks of countless seasons, and the front lines of perpetual tectonic struggle. Fly with me now, on a tour through the world of *elevation data*:



Leaflet

(These maps are interactive! Open full screen ↗ (<http://tangrams.github.io/terrain-demos/?url=styles/elevation-tiles.yaml#12/37.8773/-121.9290>))

If you see something above that looks like a lightning storm in a Gak factory, you're in the right place. This is a "heightmap" of the area around **Mount Diablo** (https://en.wikipedia.org/wiki/Mount_Diablo), about 30 miles to the east of San Francisco. The stripes correlate to constant elevations, but they're not intended to be viewed in this way – the unusual coloring is the result of the way the data is "packed" into an RGBA image: each channel encodes a different order of magnitude, combining to form a 4-digit value in base-256.

The data originates from many sources, including those compiled by the **USGS** (https://en.wikipedia.org/wiki/United_States_Geological_Survey) and released as part of **The National Map** (<http://nationalmap.gov/elevation.html>) of the United States. **Mapzen** (<http://mapzen.com>) is currently combining this data with other global datasources including ocean bathymetry, and tiling it for easy access through a tile server.

When "unpacked," processed, and displayed with WebGL, this data can be turned into what you were maybe expecting to see:

[Leaflet](#)

(*Open full screen* ↗ (<http://tangrams.github.io/terrain-demos/?url=styles/green.yaml#12/37.8773/-121.9290>))

This is a shaded terrain map, using tiled open-source elevation data, drawn in real time by your very own browser, and looking *sweet*.

We're processing this data with a view toward custom real-time hillshading, terrain maps, and other elevation-adjacent analysis, suitable for use by (for instance) the **Tangram** (<http://mapzen.com/tangram>) map-rendering library.

Why, you ask, and how? I'm glad you asked. For the Why, come with me back through time, to *the past*.

That's a Relief

Depicting terrain in a cartographic context historically required the trigonometric precision of the surveyor's craft with an artist's eye for summary and nuance, which is to say: it was difficult, time-consuming, and expensive. It was easier to just draw some bumps on the map, so your army knew which areas to avoid as it marched around conquering stuff.



1561 Girolami Ruscelli map of West Africa, [via Wikimedia](#)

(https://commons.wikimedia.org/wiki/File:1561_map_of_West_Africa_by_Girolamo_Ruscelli.JPG)

Eventually it became useful to depict the landscape as a navigable surface rather than merely a series of discrete features, though slight differences in elevation weren't seen as important; mostly people wanted to know which hills were best for putting guns on:



Thomas Hyde Page 1775 Map of Boston, *via Boston Public Library* (<http://maps.bpl.org/id/n48566>)

By contrast, true relief mapping treats elevation as a continuously variable scalar field, and depicts the terrain as though lit and viewed from above. As such, it requires comprehensive information about the area of interest.

Before satellites, acquiring this kind of information required paying meticulous attention to detail with precision equipment at high altitudes, so naturally it was pioneered by the Swiss. The first known relief map was of the Canton of Zürich, completed by **a lone, clearly-mad cartographer** (https://de.wikipedia.org/wiki/Hans_Conrad_Gyger) over 38 years. Here's a detail:



Hans Congrad Gyger 1667 Map of Canton of Zürich

The map was so good it was declared a military secret. (**Here's a link to the whole thing** (<https://commons.wikimedia.org/wiki/File:Gygerkarte.jpg>), but don't tell anybody.) By the late 1800's Switzerland had calmed down a bit and these relief techniques were publicized, resulting in a wave of very modern-looking terrain maps. Here's representative work from 1896:



Xaver Imfeld 1896 Map of Switzerland, via Heidelberg University Library (http://digi.ub.uni-heidelberg.de/diglit/grosjean_1971/0087)

Notable among relief mappers is the 20th-century Swiss master cartographer **Eduard Imhof** (http://www.library.ethz.ch/exhibit/imhof/imhof3_e.html), who drew on impressions recorded during his hikes among the Alps, including the play of light among peaks and glaciers, the density of the air at various altitudes and times of day, and the desaturation of color over great distances, to create expressive landscapes based on carefully-collected data.

Here's a detail from a map of the Swiss district of Emmental, from an old **Swiss High School Atlas** (http://www.library.ethz.ch/exhibit/imhof/imhof12_e.html), the production of which Imhof oversaw from 1932 to 1976:



And from the same volume, a detail from one of my favorite Imhof maps, of Everest:



In these maps, the hillshading serves as the base on which everything else rests: contours, elevation tinting, “hachure” lines for texture, and so on. Each of these features is a different way of describing elevation data, and each contributes to a total picture of the terrain.

Getting Off the Ground

Returning to the present, assume for sake of argument that you have this elevation data, and some way to manipulate it. How can we get from there to maps such as the ones seen above?

Even in its raw, Gak-like form, elevation data is immediately useful for creating terrain maps. Using soon-to-be-released **Tangram** (<http://mapzen.com/tangram>) functionality, we can apply the elevation data to our map tiles as a texture, and then use a custom shader to generate a “hypsometric” map, which applies a color gradient to the unpacked elevation range. Here’s one with a simple grayscale:

(Open full screen ↗ (<http://tangrams.github.io/terrain-demos/?url=styles/grayscale.yaml#12/37.8773/-121.9290>))

The classic hypsometric tinting scheme has green in the lower elevations, white on higher elevations, and earth tones in between. It's generally frowned upon because it starts to look meaningful, as though we were using color to represent some other kind of data such as landcover. Here's a typical example:

(*Open full screen ↗* (<http://tangrams.github.io/terrain-demos/?url=styles/hypsometric.yaml#12/37.8773/-121.9290>))

With a more complex gradient, you can make a gratuitously animated contour map:

(*Open full screen ↗* (<http://tangrams.github.io/terrain-demos/?url=styles/contours.yaml#12/37.8773/-121.9290>))

But for proper hillshading, we need to light our terrain. To do that, we need more information.

The New Normal

Though a Tangram map is technically a 3D scene, our map tiles are simple flat squares, and flat light on a flat surface produces a flat map. To represent terrain in a lightable way, we need bumpy tiles. It's possible to add geometric detail to the tiles, but such high-resolution data would require very high-resolution geometry, which as of this writing requires very high-end hardware to process and display in real-time. So let's fake it instead.

In mathematics, the perpendicular of a tangent to a curve is called the “normal.” The concept extends to three directions – you can think of a 3D normal as the direction a 3D face “faces.” And if you have the position and normal of a three-dimensional face, you can tell how it ought to reflect light.

Here’s the trick: in 3D scenes, it’s possible to manipulate the normal of a face directly, which changes its appearance *as though* the geometry were changing. I know, I know. Right? But wait, there’s more!

Through the magic of WebGL we have access to every pixel on any 3D face, so we can manipulate the normal of a face *per-pixel*, which is called “**normal mapping** (https://en.wikipedia.org/wiki/Normal_mapping).” This is an exceedingly cool trick, and is used everywhere in video games and film VFX to very cheaply add apparent detail to 3D objects.

We can compute the normal of each pixel of a heightmap by comparing that pixel with its neighbors. The result is a normal map, which gives us the “slope” of the height data at each pixel. This can be done live in a shader with texture lookups, but it’s simpler to calculate these maps offline and serve them separately, and Mapzen has a normal map tile server doing just that. Here’s what those tiles look like:

(Open full screen ↗ (<http://tangrams.github.io/terrain-demos/?url=styles/normals-tiles.yaml#12/37.8773/-121.9290>))

Here, the three RGB channels are mapped to the axes of 3D space, with a lower-resolution heightmap in the alpha just for fun. Once this normal map is applied, our previously flat elevation data pops with 3D texture and responds to light like any other 3D object, yet it still plays nice underneath other map data.

Let There Be Lights

Now that we have normals, we can add a light to the scene. However, with a single light, surfaces which don’t face the light are lumped together in shadow. You can get a sense of relative elevation, but it’s not always obvious which features are more prominent overall, and the primary effect is an ambiguous sense of texture. Here’s a scene with a single directional light at an extreme angle to illustrate the point:

(*Open full screen ↗* (<http://tangrams.github.io/terrain-demos/?url=styles/single-light.yaml#12/37.8773/-121.9290>))

While generally undesirable, we can exploit this phenomenon by pointing our light straight down at the terrain, brightening flat areas. This immediately produces the classic “slopesshade” often used as a layer in topographic maps:

(*Open full screen ↗* (<http://tangrams.github.io/terrain-demos/?url=styles/slope.yaml#12/37.8773/-121.9290>))

But if your goal is a hillshade, two light sources can cover different sets of angles, allowing for a kind of luminous stereoscopic triangulation. Using only two directional lights, we can immediately recreate classic hillshading looks, with greater depth and more subtle shading:

(Open full screen ↗ (<http://tangrams.github.io/terrain-demos/?url=styles/two-lights.yaml#12/37.8773/-121.9290>))

But why stop there? The classic theatrical and studio photographic lighting setup is made of three lights – key, fill, and rim – and allow the subject to be separated from and grounded in its context. In our case there isn't a background *per se*, but we can use similar principles to suggest sky light with a warm key and a cool fill, and highlight smaller, steeper features with a third, low-angle kicker:

(Open full screen ↗ (<http://tangrams.github.io/terrain-demos/?url=styles/three-lights.yaml#12/37.8773/-121.9290>))

As we know from strings of holiday lights, the more light sources which contribute to a scene, the softer the resulting light, and the sharper an angle needs to be to stand out. And in general, multiple colors of light, illuminating the same feature from multiple angles, give us the best chance to determine the shapes of solid objects.

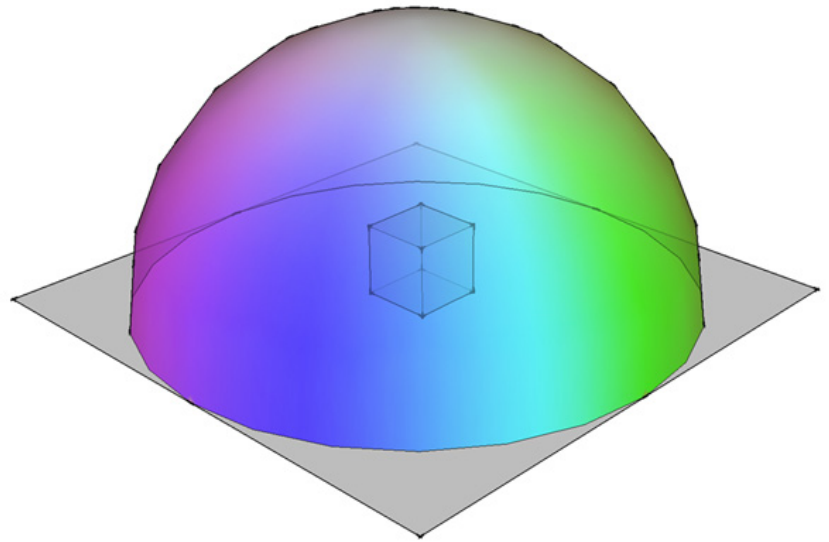
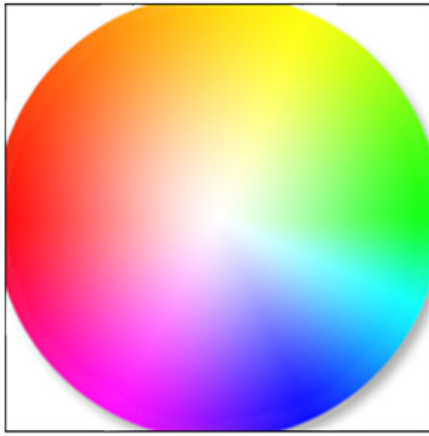
Not coincidentally, soft, multi-colored light is our typical daily experience, as light reaches us both directly from light sources and indirectly, reflected from surfaces all around us. We're well-adapted to this situation, and with this kind of light mix, our brains reliably provide us with the sight of solid, well-formed objects in space.

Ideally, we could replicate this situation to produce a good-looking hillshade. However, multiple lights are expensive in graphics terms – each light is an extra set of calculations which must be factored into the overall lighting equation. So you *could* add ever more lights to build up a complex lighting environment, but at the cost of performance.

Sphere Goes Nothing

Luckily, there's a graphics trick that can simulate a whole skyfull of lights, known as an environment map. (The variant we're using here is called a spherical environment map – in the docs **we call them spheremaps** (<https://github.com/tangrams/tangram-docs/blob/gh-pages/pages/Materials-Overview.md#mapping-spheremap>).)

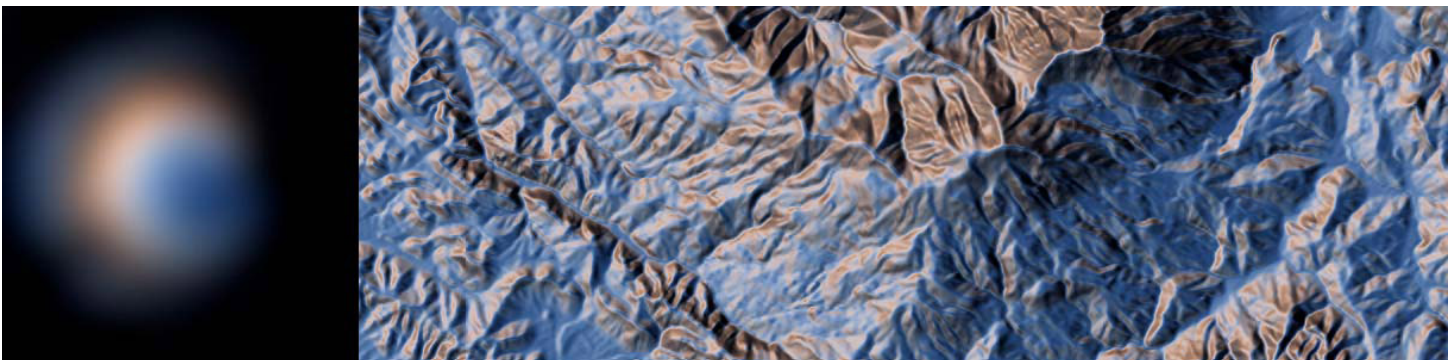
With this technique, a source image is “stretched” over an infinitely large hemisphere covering the scene. Then, each face of the 3D geometry (or each pixel in the normal map) is assigned a new color according to which part of the source image it “faces.”



The effect is of a strongly colored sky casting light on the scene, which can produce striking results with relatively little calculation. For our purposes, smooth gradients in limited palettes work best. Here are some examples, to give you the idea (click each for a full-screen interactive version):



(<https://tangrams.github.io/terrain-demos/?url=styles/environment-map1.yaml#12/37.8773/-121.9290>)



(<https://tangrams.github.io/terrain-demos/?url=styles/sunrise.yaml#12/37.8773/-121.9290>)

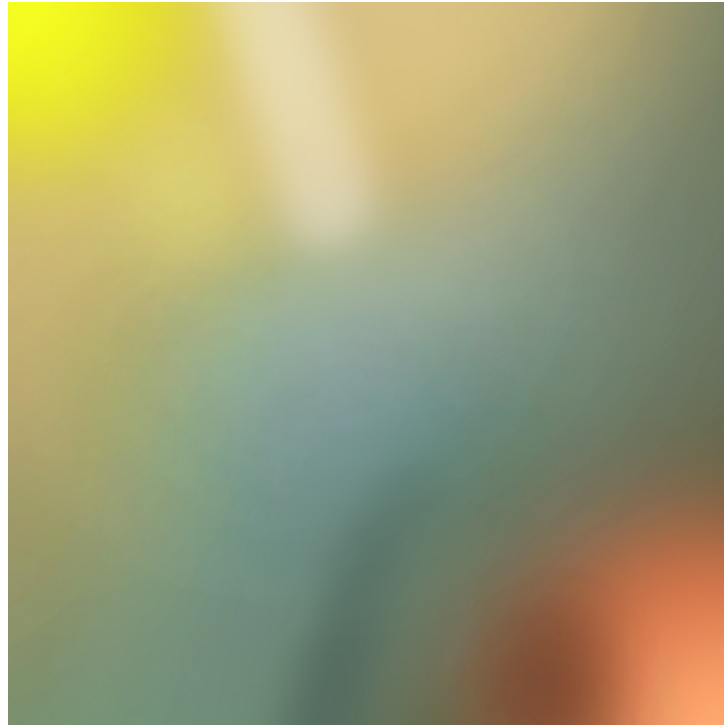


(<https://tangrams.github.io/terrain-demos/?url=styles/sunset.yaml#12/37.8773/-121.9290>)



(<https://tangrams.github.io/terrain-demos/?url=styles/metal.yaml#12/37.8773/-121.9290>)

And at the risk of being struck dead for hubris by the ghost of Eduard Imhof, here's a hastily-daubed homage to the Swiss Style:



(*Open full screen* ↗ (<http://tangrams.github.io/terrain-demos/?url=styles/imhof.yaml#12/37.8773/-121.9290>))

Throwing Shade

The main pitfall with hillshading is that it begins to look realistic. If a mountain *could* be that color, it looks as though the data says it *is* that color. It's very easy to cross the conceptual boundary from “requires interpretation” to “looks plausible,” past which your forebrain shuts off. But navigating that boundary is one of the things that makes this work so much fun.

Terrain mapping is not a straightforward or exact science, and the shaders demonstrated here are no replacement for a sensitive and thoughtful eye. And of course, hillshading is really only the start of what can be done with this kind of data. These examples are just gestures, pointing toward new ways of seeing our most basic and overlooked contexts.

We're going to continue working on real-time terrain mapping and analysis techniques, and we're very interested to see what happens when more people have access to this data directly. We plan to make the tilesets public soon, starting with California, and then – the world!

(The code for all of these maps is available **on Github** (<http://github.com/tangrams/terrain-demos>))



Test Pilots Wanted

Give our terrain tiles a spin with these tile endpoints and **learn more about them here** (<https://mapzen.com/documentation/terrain-tiles/>):

- <https://tile.mapzen.com/mapzen/terrain/v1/terrarium/{z}/{x}/{y}.png>
(<https://tile.mapzen.com/mapzen/terrain/v1/terrarium/%7Bz%7D/%7Bx%7D/%7By%7D.png>)
- <https://tile.mapzen.com/mapzen/terrain/v1/normal/{z}/{x}/{y}.png>
(<https://tile.mapzen.com/mapzen/terrain/v1/normal/%7Bz%7D/%7Bx%7D/%7By%7D.png>)
- <https://tile.mapzen.com/mapzen/terrain/v1/geotiff/{z}/{x}/{y}.tif>
(<https://tile.mapzen.com/mapzen/terrain/v1/geotiff/%7Bz%7D/%7Bx%7D/%7By%7D.tif>)
- <https://tile.mapzen.com/mapzen/terrain/v1/skadi/{N|S}{y}/{N|S}{y}{E|W}{x}.hgt.gz>
(<https://tile.mapzen.com/mapzen/terrain/v1/skadi/%7BN%7CS%7D%7By%7D/%7BN%7CS%7D%7By%7D%7BE%7CW%7D%7Bx%7D.hgt.gz>)

Terrarium format PNG tiles contain raw elevation data in meters. All values are positive with a 32,768 offset, split into the red, green, and blue channels, with 16 bits of integer and 8 bits of fraction. To decode:

```
(red * 256 + green + blue / 256) - 32768
```

Normal format PNG tiles are processed elevation data with the the red, green, and blue values corresponding to the direction the pixel “surface” is facing (its XYZ vector). The alpha channel contains quantized elevation data with values suitable for common hypsometric tint ranges. High alpha channel values indicate lower elevation values (below sea level), making them more opaque.

Specifically, normal format alpha values are counted in (floored) elevation increments. Below sea level they start at -11,000 meters (Mariana Trench) and range to -1,000 meters in 1,000 meter increments, with more detail on the coastal shelf at -100, -50, -20, -10 and -1 meters and finally 0 (intertidal zone). Values above sea level are reported in 20 meter increments to 3,000 meters, then 50 meter increments until 6,000 meters, and then 100 meter increments until 8,900 meters (Mount Everest).

For the moment these tiles cover California only, but worldwide coverage is coming in the months ahead. In the meantime, to query elevation at a single point or along a path almost anywhere in the globe, **sign up for an API key** (<https://mapzen.com/developers/>) and check out the **Mapzen Elevation Service** (<https://mapzen.com/documentation/elevation/>).

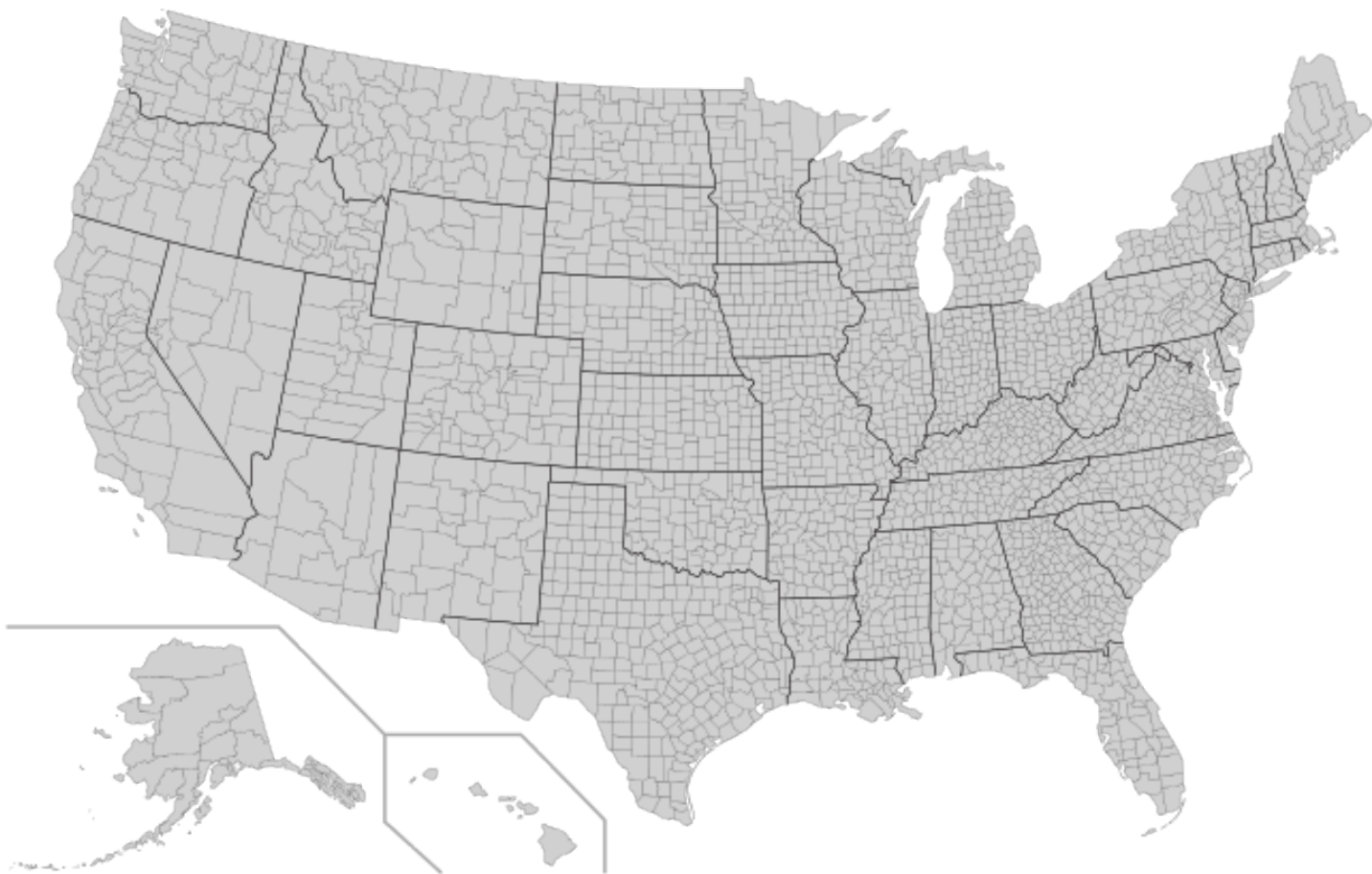
Have any feedback on the tiles? Please direct it to **Nathaniel Kelso** (<https://github.com/nvkelso>), our Head of Data & Cartography, at **hello@mapzen.com** (mailto:hello@mapzen.com)!

UPDATE Jan 19, 2017: we've deprecated the terrain-preview beta endpoints and have updated this blog with the proper URLs

Not all counties are equal

[search \(/tag/search\)](/tag/search) [geocoding \(/tag/geocoding\)](/tag/geocoding)

Welcome to a new series, Administrivia: The amazing structure of the places we live.



*US Counties, via **via Wikimedia***

(https://en.wikipedia.org/wiki/List_of_United_States_counties_and_county_equivalents#/media/File:Usa_counties_large.svg)

Counties are something to be appreciated. Their names resonate. Americans know the county they live in, they know they pay taxes there. Annual **county fairs**

(https://en.wikipedia.org/wiki/County_fair) are a time-honored tradition in many counties, some (<http://yorkfair.org/our-history/>) that predate the United States.

It's names. It's organizations. It's history (e.g. Louisiana church parishes). It's organization of Alaska (where counties were forbidden). It's the history of independent cities (why did these folks break away?). This is less of geocoding blog post and more of an appreciation of administrative trivia. These are the crazy levels of democracy.

Most countries on this planet we call **Earth**

(<https://whosonfirst.mapzen.com/spelunker/search/?q=earth&placetype=planet>) have administrative breakdowns to handle things like elected representatives, taxation, and law enforcement. In the United States with a population of over 320 million, the first subdivision consists of states, territories, and a federal district (this is a bit simplistic, but for the sake of brevity I'm going to generalize a bit). States are composed of counties, which are, in turn, composed of cities. **Lichtenstein** (<https://whosonfirst.mapzen.com/spelunker/id/85633267/>), however, with 0.002% of the area and 0.012% of the population of the US, requires less complexity to administer effectively.

In the United States, we tend to call these "administrative subdivisions of a state" *counties*.

Examples include **Lancaster County**

(<https://whosonfirst.mapzen.com/spelunker/id/102081377/>) in Pennsylvania and **Wayne County** (<https://whosonfirst.mapzen.com/spelunker/id/102083147/>) in Michigan. The island of Maui in Hawaii is 1 of 5 islands that comprise **Maui County** (<https://whosonfirst.mapzen.com/spelunker/id/102085577/>). The word 'County' isn't just decoration, it's part of the official name.

If the gazetteer for your geocoder doesn't include the fully-qualified name, it's quite understandable to assume that appending the word 'County' would suffice, but what appears to be a simple rule reveals a much more complicated underbelly upon closer inspection.

County Seats

The center of administration for a county is a city within the county called a "county seat". In many cases, the county seat is a city with the same name as the county and may not be the most populous city in the county. For example, the city of **Socorro, NM** (<https://whosonfirst.mapzen.com/spelunker/id/85976677/>) is the county seat of **Socorro County, NM** (<https://whosonfirst.mapzen.com/spelunker/id/102081575/>). That's not always the case, though, as the county seat of **White Pine County, NV** (<https://whosonfirst.mapzen.com/spelunker/id/102081551/>) is **Ely, NV** (<https://whosonfirst.mapzen.com/spelunker/id/85974771/>).

Some counties, such as **Essex County, MA**

(<https://whosonfirst.mapzen.com/spelunker/id/102084463/>) and **Tallahatchie, MS**

(<https://whosonfirst.mapzen.com/spelunker/id/102085903/>), have 2 county seats.

County Names

It's a forgivable peccadillo to assume that all county names end with the word "County". This is true for the vast majority of the 3144 county-level governments in the United States, but history has a few things to say along the way.

Louisiana

The area that eventually became **Louisiana**

(<https://whosonfirst.mapzen.com/spelunker/id/85688735/>), once under the oversight of French and Spanish Roman Catholic church, was made up of administrative territories that roughly coincided with church parishes. The name stuck so, instead of being given the "County" name, every county-level governmental entity in the state ends with "Parish". Some examples are **East Baton Rouge Parish** (<https://whosonfirst.mapzen.com/spelunker/id/102087229/>) and **Orleans Parish** (<https://whosonfirst.mapzen.com/spelunker/id/102086693/>) – there are **64 parishes** (<https://whosonfirst.mapzen.com/spelunker/id/85688735/descendants/?&placetype=county>) all together.

Alaska

Alaska (<https://whosonfirst.mapzen.com/spelunker/id/85688573/>) is broken down administratively into 20 "boroughs". Of the 20 boroughs, there are 19 organized boroughs while the remaining land in the state is organized into 10 "census areas" (collectively commonly referred to as the "unorganized borough"). An organized borough contains a borough seat (equivalent to a county seat), while the census areas have no city or town to serve as their administrative centers.

Six of the Alaskan boroughs, referred to as "consolidated city-boroughs", do not further break down administratively into cities. That is, the governments of the boroughs of **Anchorage** (<https://whosonfirst.mapzen.com/spelunker/id/102085881/>), **Haines** (<https://whosonfirst.mapzen.com/spelunker/id/102083767/>), **Juneau** (<https://whosonfirst.mapzen.com/spelunker/id/102081399/>), **Sitka** (<https://whosonfirst.mapzen.com/spelunker/id/102081405/>), **Wrangell** (<https://whosonfirst.mapzen.com/spelunker/id/102081901/>), and **Yakutat** (<https://whosonfirst.mapzen.com/spelunker/id/102081907/>) function as the centers of

government for any cities and towns contained in those boroughs. Other boroughs, such as **Denali** (<https://whosonfirst.mapzen.com/spelunker/id/102081907/>) consist of cities and towns that can have their own local government.

While boroughs in Alaska are administratively classified as such, their names aren't as simple as appending 'Borough', their names are quite varied. Some examples are:

1. **Kodiak Island Borough** (<https://whosonfirst.mapzen.com/spelunker/id/102081005/>)
2. **Juneau City and Borough** (<https://whosonfirst.mapzen.com/spelunker/id/102081399/>)
3. **Municipality of Skagway** (<https://whosonfirst.mapzen.com/spelunker/id/102081007/>)

For naming purposes, the census areas are all appended with "Census Area", for example:

1. **Dillingham Census Area** (<https://whosonfirst.mapzen.com/spelunker/id/102080995/>)
2. **Nome Census Area** (<https://whosonfirst.mapzen.com/spelunker/id/102081401/>)
3. **Yukon-Koyukuk Census Area**
(<https://whosonfirst.mapzen.com/spelunker/id/102081905/>)

Why were they called boroughs? In 1912, when Alaska became a U.S. territory, Congress **forbade the Alaskan Government from making counties** (<http://www.akhistorycourse.org/articles/article.php?artID=135>) (along with doing a lot of other things (http://www.legis.state.ak.us/basis/get_documents.asp?session=27&docid=12807)).

Independent Cities

Independent cities in the United States are cities that belong to no county but are treated as counties for governing purposes. That is, whereas a typical county consists of myriad cities and towns with the county acting as the "parent" government, independent cities are treated as an immediate descendant of a state.

While **Virginia** (<https://whosonfirst.mapzen.com/spelunker/id/85688747/>) has 38 independent cities (such as **Virginia Beach** (<https://whosonfirst.mapzen.com/spelunker/id/101728745/>) and **Poquoson** (<https://whosonfirst.mapzen.com/spelunker/id/101728893/>)), there are only 3 others in the United States:

- **Baltimore, MD** (<https://whosonfirst.mapzen.com/spelunker/id/85949461/>)
- **Carson City, NV** (<https://whosonfirst.mapzen.com/spelunker/id/102082603/>)
- **Saint Louis, MO** (<https://whosonfirst.mapzen.com/spelunker/id/102080791/>)

Where things get a bit confusing is that the independent cities of **Baltimore, MD** (<https://whosonfirst.mapzen.com/spelunker/id/85949461/>) and **St. Louis, MO** (<https://whosonfirst.mapzen.com/spelunker/id/85971343/>) share borders with (but are not part of) **Baltimore County, MD** (<https://whosonfirst.mapzen.com/spelunker/id/102081907/>) and **St. Louis County, MO** (<https://whosonfirst.mapzen.com/spelunker/id/102086173/>), respectively.

Combined City-County Governments

Unlike independent cities, some city and county governments are one and the same. Notable examples include **San Francisco** (<https://whosonfirst.mapzen.com/spelunker/id/85922583/>), **Denver** (<https://whosonfirst.mapzen.com/spelunker/id/85928879/>), **Philadelphia** (<https://whosonfirst.mapzen.com/spelunker/id/101718083/>), and **Honolulu** (<https://whosonfirst.mapzen.com/spelunker/id/102085417/>) (along with **Anchorage** (<https://whosonfirst.mapzen.com/spelunker/id/85915707/>) and **New Orleans** (<https://whosonfirst.mapzen.com/spelunker/id/85948111/>)). **San Francisco County** (<https://whosonfirst.mapzen.com/spelunker/id/102087579/>) once extended down what is now **Palo Alto** (<https://whosonfirst.mapzen.com/spelunker/id/85922351/>), but was **broken in two** (<https://whosonfirst.mapzen.com/spelunker/id/102085387/>) in 1856 by the California state legislature. The city and county governments were merged, **nominally to fight city corruption and boom town violence** (<http://www.trampsofsanfrancisco.com/defining-san-francisco-how-our-city-became-a-city-part-iii/>).



California Counties in 1856, **via David Rumsey**

(<http://www.davidrumsey.com/luna/servlet/detail/RUMSEY~8~1~1700~130059:California---with--City-of-San-Fran>)

What Counties Do

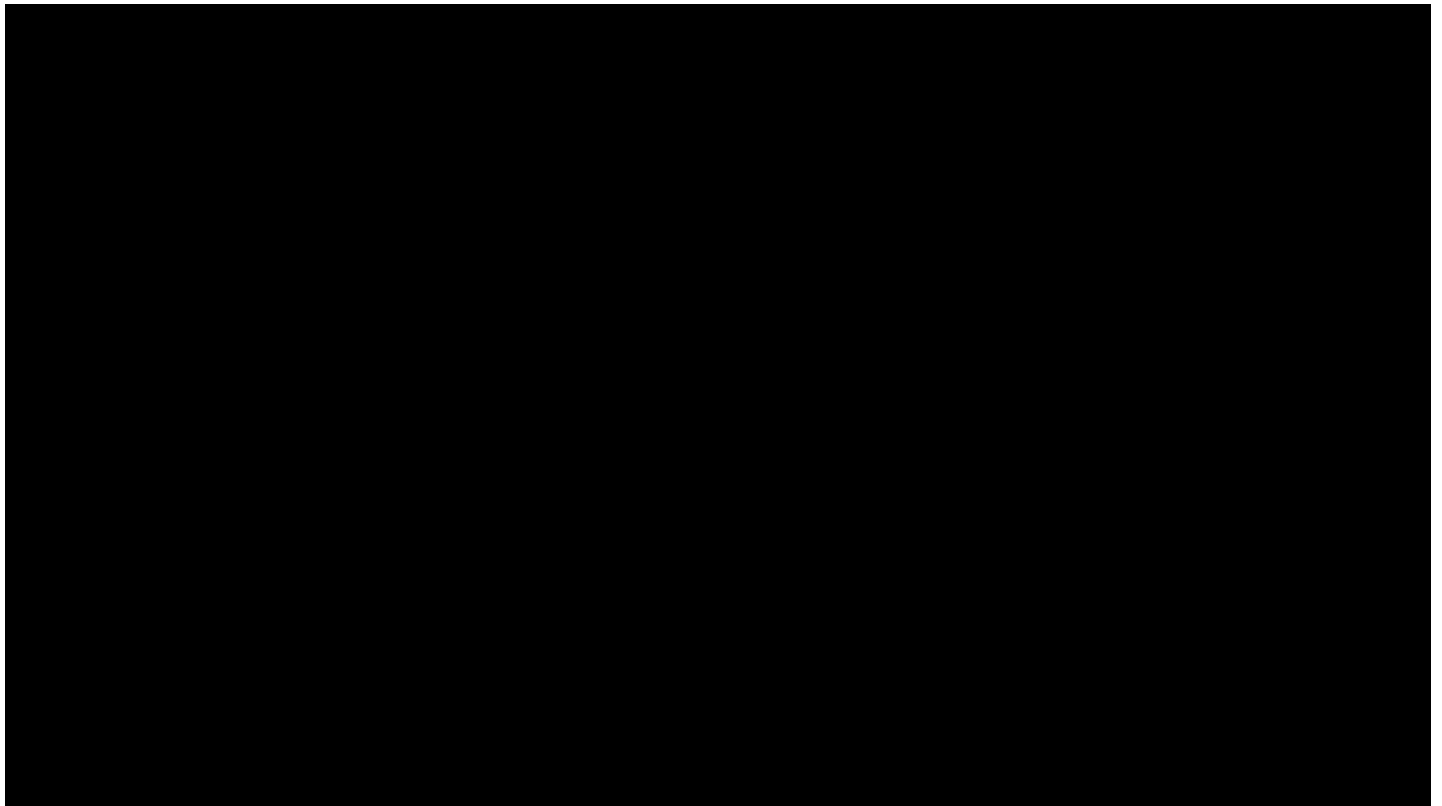
Unless you live in Connecticut or Rhode Island¹, which have mostly eliminated the governmental functions of counties, your county probably provides a whole host of functions you probably don't think about on a regular basis, but makes up a ton of what we think of as local government, but the scope of their function varies tremendously state-by-state.

Counties are usually responsible for local courts and have law enforcement (in the form of the County Sheriff). In most places they've got the power to tax, issue government bonds, can coordinate local infrastructure projects, and run local services like schools, libraries, jails, and governmental records offices. Often they're responsible for organizing elections.

And while not the case in all areas, the wide-ranging power of counties can have a dramatic impact on its citizens lives.

The Allegory of the “Special District”

An interesting way to look at the power of counties is not through a county itself, but a “Special District”, a form of local government that's created for a specific purpose and holds many of the same powers as counties. Last Week Tonight with John Oliver has a particularly great overview of special districts.



But perhaps the most famous special district is the **Reedy Creek Improvement District** (https://en.wikipedia.org/wiki/Reedy_Creek_Improvement_District) outside **Orlando, Florida** (<https://whosonfirst.mapzen.com/spelunker/id/85933091/>). It exists as the local government for Disney World. It's made up of land that's part of Florida's **Orange** (<https://whosonfirst.mapzen.com/spelunker/id/102085847/>) and **Osceola** (<https://whosonfirst.mapzen.com/spelunker/id/102085819/>) counties, but for all intents and purposes, it's the county. It's got an elected Board of Supervisors, but as the Orlando Sentinel

noted in a 2011 profile² of a rare election in the district, landowners in the district get one vote per acre of land they own and it's Disney that owns 2/3 of the land. In order to be elected to the board, you have to be a landowner as well, so incoming board members are given 5 inaccessible, undevelopable acres to qualify, but then sign a contract that requires them to give back the land when their time on the board ends. (Rest assured, most counties don't give their elected land to qualify for administration).

With the power of a special district comes the power to charge taxes (primarily sales taxes), issue public bonds for the construction of infrastructure, run and repair roads, canals, utilities, and run the fire department. They set the zoning rules, building codes, and environmental control rules.

There are 2 full cities, **Lake Buena Vista**

(<https://whosonfirst.mapzen.com/spelunker/id/85933109/>) and **Bay Lake**

(<https://whosonfirst.mapzen.com/spelunker/id/85933129/>), which are unincorporated to either of the counties, but have their own Mayors and City Councils (also elected by landowners) who are already employees of Disney World. And when Disney created the planned town of Celebration, Florida they de-annexed the territory of the town back into the exclusive jurisdiction of Osceola County so it wouldn't have to get involved in the affairs of actual residents who weren't employees of the park.

And while there's neither law enforcement capacity nor courts, it doesn't stop people asking about **Disney Jail** (<http://www.disneyquestions.com/really-thing-disney-jail-disney-parks/>).

It's a remarkable microcosm of local government³. The choices of what Reedy Creek Improvement District does, and importantly, doesn't do show an awful lot about where the powers of individual companies and people end and government begins.

Celebrate Counties

And so we conclude our brief tour of counties. There's a lot more wonderful and weirdness to catalog. If this excites you, you might want to take a look at our open data project for organizing the world, **Who's on First** (<https://whosonfirst.mapzen.com>) and consider getting involved.

U.S. counties may be weird, but they are ours, a combination of countless decisions, made by known and unknown citizens, in **every state** (<http://www.mapofus.org/iowa/>), over nearly **four centuries** (<http://flowingdata.com/2015/05/15/animated-history-of-us-county-boundaries/>).



(You can dig into historic county boundary data **via the Newberry Library** (http://publications.newberry.org/ahcbp/downloads/united_states.html) if you want to make your own visualizations.)

If you want to dig deeper into counties, the Wikipedia article ***County (United States)*** ([https://en.wikipedia.org/wiki/County_\(United_States\)](https://en.wikipedia.org/wiki/County_(United_States))) is a great place to start. This post owes a great debt of gratitude to the amazing Wikipedia article on the **Reedy Creek Improvement District** (https://en.wikipedia.org/wiki/Reedy_Creek_Improvement_District) as well as the book ***Married to the Mouse: Walt Disney World and Orlando*** (https://openlibrary.org/books/OL9634791M/Married_to_the_Mouse) by Richard Foglesong.

And if you have opinions about counties, or what we wrote about them, **let us know over in this gist** (<https://gist.github.com/trescube/92a1037e564bc72ec8c4>)!

-
1. If you're not in the United States, fair, you're probably also not in a county. Same too with many of the US's territories, which have other administrative districts between the city and the territory. ↩

2. Orlando Sentinel - Disney's Reedy Creek government has a rare board vacancy, but don't bother running, http://articles.orlandosentinel.com/2011-05-09/business/os-disney-reedy-creek-election-20110509_1_disney-awards-reedy-creek-improvement-district-tom-moses (http://articles.orlandosentinel.com/2011-05-09/business/os-disney-reedy-creek-election-20110509_1_disney-awards-reedy-creek-improvement-district-tom-moses) ↩
3. Something something, **William Gibson** (<http://www.wired.com/1993/04/gibson-2/>) ↩

· 23 March 2016 ·



Stephen Hess

Stephen works on geocoding exclusively as a means to fund his passion for designing and building wooden frames for old maps.



David Riordan

Former product manager for Mapzen Search. Historical mapping, open tools, open access, and open data.



John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

How we manage our APIs with ApiAxe

engineering (/tag/engineering) **demo** (/tag/demo)

In the fourth post in our **Engineering at Mapzen series** (<https://mapzen.com/tag/engineering>), we dig into API management and rate limiting using ApiAxe with a demo in Docker.

Don't let your APIs manage you

Say you need to manage access to an API. Direct access to the underlying system usually needs to be restricted, and you may want to meter and measure usage for a particular group of users identified by an API token. We do this using **ApiAxe** (<https://github.com/apiaxe/apiaxe>), an open source proxy tool, and wanted to share what we learned. But we're not just going to throw words at you – this post doubles as a working tutorial that you can follow along with using **Docker** (<https://www.docker.com>). We talk more about why we chose ApiAxe **at the end of this post**.

Terminology

For the sake of clarity, let's get some of the terminology out of the way:

- **API** (application program interface) is a set of routines, protocols, and tools for building software applications. In our context they allow developers to use search, turn-by-turn and vector tiles in their applications.
- A **proxy** is a software layer that runs between a client and the service origin.

Here's the software we will use in the demo:

- **ApiAxe** (<https://github.com/apiaxe/apiaxe>) is a proxy that sits on your network, in front of your API(s) and manages things that you shouldn't have to, like rate limiting, authentication and analytics. It's fast, open and easy to configure.
- **Docker** (<https://www.docker.com>) is a technology that manages LXC (Linux containers). This Linux kernel technology creates lightweight virtualized OS environments using cgroups and namespaces rather than emulating the hardware layer. This allows you to create many

of them without it being a drag on the host system.

- **Docker compose** (<https://docs.docker.com/compose/>) is software for defining and running multicontainer Docker applications. With it, you use a `docker-compose` file to configure your application's tasks. Then, using a single command, you can create and start all the tasks from the configuration file.
- **Docker Hub** (<https://hub.docker.com>) is a cloud-hosted service from Docker that provides registry capabilities for public and private content and supports collaboration. (Think of it as GitHub for Docker images.)
- **Redis** (<http://redis.io>) is an open-source, in-memory data structure store, used as database, cache and message broker.
- **Node.js** (<https://nodejs.org>) is an open-source, cross-platform runtime environment for developing server-side web applications using JavaScript.
- **nginx** (<https://www.nginx.com>) is a popular open-source web server.

Now for the exciting bit...

If you are unfamiliar with Docker, it is a very exciting way of working with complex multihost distributed systems on your local development environment. It is ideal for this kind of demonstration, though it might be frustrating if you haven't worked with it before. But you can do it! Installation instructions are **here** (<https://docs.docker.com/engine/installation/>).

The material was made and tested with the following versions of docker and docker-compose:

```
→ ~ docker -v
Docker version 1.8.2, build 0a8c2e3
→ ~ docker-compose -v
docker-compose version: 1.4.2
```

If you like to follow along... (and my apologies in advance for using a Git submodule – I just don't dislike them as much as many developers do.)

```
git clone --recursive git@github.com:mapzen/docker-apiaxle-demo.git
cd docker-apiaxle-demo
```

This repository has some content in the **readme** (<https://github.com/mapzen/docker-apiaxe-demo/blob/master/README.md>) but these will be more step-by-step instructions, annotated with pictures.

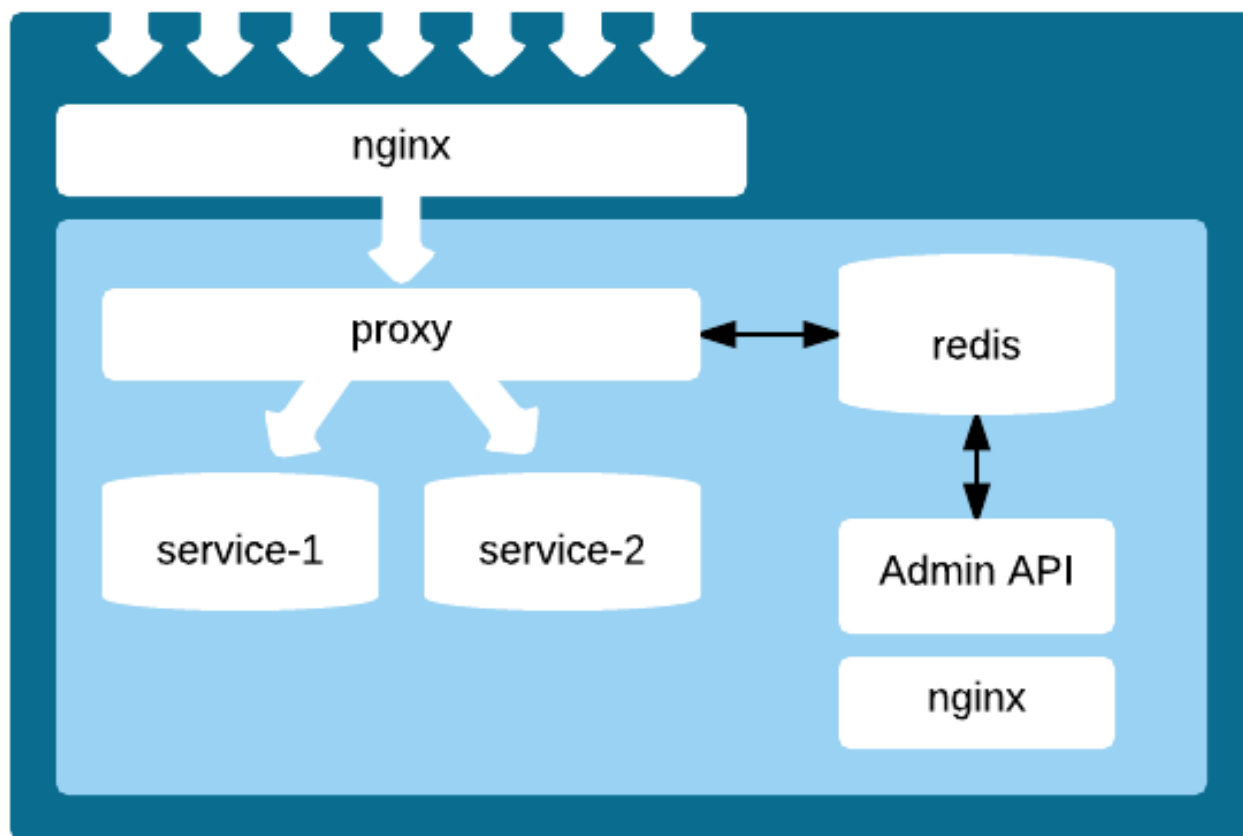
Once you have the code take a look at the **docker-compose.yml** (<https://github.com/mapzen/docker-apiaxe-demo/blob/master/docker-compose.yml>) file. This lists all the nodes in the system that you'll boot up on your machine. This includes Redis, API (for managing ApiAxle), web-proxy (NGINX frontend for the proxy), proxy (ApiAxle), proxy-event-subscriber (ApiAxle's way of subscribing to Redis events for stats), fake-services (two APIs we're going to manage with ApiAxle). This might be a lot to take at once but it will make sense towards the end.

To get started we have to build and run the images locally. There is one image that I hosted on Docker Hub, which will save you from building it on your own. You can find the Dockerfile for it **here** (<https://github.com/mapzen/docker-apiaxe-demo/blob/master/Dockerfile>).

```
docker-compose build
docker-compose up
```

If everything goes according to the plan you should see a streaming stdout from all the services defined in the docker-compose.yml file. Now we're finally ready to get started.

The following illustrations show the setup for this tutorial. While this example bundles everything in a single setup, in reality there are two steps: in addition to the administrative API, there would be an admin app which would never be publicly exposed and would reside behind a different frontend.



The top part of the diagram shows the API proxy where the users are routed through to the right service. This setup is a vanilla NGINX setup that listens for configured domain names.

```
server {  
    listen 80;  
    server_name service-1;  
    location / {  
        proxy_pass http://cluster;  
        proxy_set_header Host "service-1.api.localhost";  
    }  
}
```

This will also set the host header for the ApiAxle proxy software so that it knows how to dispatch it. The proxy software expects the hostname to have “.api.” appended to it. The name is used for lookup and you can add a suffix such as localhost to indicate that it’s running locally.

Before we start you need to add the following into your `/etc/hosts` file ... if you're on a Mac you'll need to switch `127.0.0.1` with the value of the Docker host on your system.

```
127.0.0.1 apiaxle-api service-1 service-2
```

For this to work the API needs to be registered with ApiAxle. This can be done in two ways – one is through the API:

```
curl -H 'content-type: application/json' \
-X POST \
-d '{"endPoint":"fake-services:8001"}' \
'http://apiaxle-api/v1/api/service-1'
```

```
{
  "meta": {
    "status_code": 200,
    "version": 1
  },
  "results": {
    "allowKeylessUse": false,
    "corsEnabled": false,
    "disabled": false,
    "endPoint": "fake-services:8001",
    "endPointTimeout": 2,
    "keylessQpd": 172800,
    "keylessQps": 2,
    ...
  }
}
```

API for the ApiAxle

With the API we can perform basic CRUD operations on the projects and also mint API keys for projects, link them to projects, and set limits on those keys. The API is documented **here** (<http://apiaxle.com/api.html>).

Frontend to work with the API for ApiAxle

The frontend is a very simple tool we cobbled together as we were learning the API for ApiAxle. This will allow you to read project stats and see them as you start using it. (It will not do any key minting or linking – to do that we'll follow the readme instructions on the project.)

Now let's go step by step

Fire up the environment

```
docker-compose build
docker-compose up
```

Curl into the API (we'll only use the API here, not the web admin)

```
curl apiaxle-api/v1/apis
```

If your response is...

```
{
  "meta": {
    "pagination": {
      "next": {},
      "prev": {}
    },
    "status_code": 200,
    "version": 1
  },
  "results": []
}
```

...that means we're ready to move on to the next steps.

Try hitting the the `service-1` or `service-2` endpoints:

```
curl service-1
```

And you'll get ApiUnknown error back:

```
{
  "meta": {
    "status_code": 404,
    "version": 1
  },
  "results": {
    "error": {
      "message": "'service-1' is not known to us.",
      "type": "ApiUnknown"
    }
  }
}
```

This just means that we haven't created an API. Let's create both `service-1` and `service-2` in slightly different ways.

For the first one we'll allow what is called a keyless access, where no key is required to access the service. Under the hood a temporary key is created based on the IP address. Note that rate limiting is based on it.

```
curl -H 'content-type: application/json' \
-X POST \
-d '{"endPoint":"fake-services:8001", "allowKeylessUse": true}' \
'http://apiaxle-api/v1/api/service-1'
```

Now if you try to access the service-1,

```
curl -v service-1

....

X-ApiaxleProxy-Qps-Left: 1
X-ApiaxleProxy-Qpd-Left: 172798
```

you'll see that the ApiAxle will return headers that indicate the remaining limits on your service. Note that this does not require a key.

Now let's do the same for `service-2`, but require a key for its use.

```
curl -H 'content-type: application/json' \  
-X POST \  
-d '{"endPoint":"fake-services:8002", "allowKeylessUse": false}' \  
'http://apiaxle-api/v1/api/service-2'
```

if you try to access this API now you'll get 403 Forbidden .

```
curl -v service-2  
  
...  
  
{  
  "meta": {  
    "status_code": 403,  
    "version": 1  
  },  
  "results": {  
    "error": {  
      "message": "No api_key specified.",  
      "type": "KeyError"  
    }  
  }  
}
```

We can address this by creating a key:

```
curl -H 'content-type: application/json' \
  -X POST \
  -d '{}' \
  'http://apiaxle-api/v1/key/something-special'

...

{
  "meta": {
    "status_code": 200,
    "version": 1
  },
  "results": {
    "createdAt": 1457984190955,
    "disabled": false,
    "qpd": 172800,
    "qps": 2
  }
}
```

And then we link to the `service-2` API:

```
curl -H 'content-type: application/json' \
  -X PUT \
  'http://localhost/v1/api/service-2/linkkey/something-special'
```

Now we can call the `service-2` API:

```
curl -v 'service-2?api_key=something-special'
...
X-ApiaxleProxy-Qps-Left: 1
X-ApiaxleProxy-Qpd-Left: 172799
```

And now if we create yet another key with very low limits, we can see what the response is when key limits are exceeded.

```
curl -H 'content-type: application/json' \
  -X POST \
  -d '{"qpd": 1}' \
  'http://apiaxle-api/v1/key/almost-no-requests-per-day'

curl -H 'content-type: application/json' \
  -X PUT \
  'http://localhost/v1/api/service-2/linkkey/almost-no-requests-per-day'
```

We only need to fire off two calls to `service-2` and we should see the limit hit.

```
curl -v 'service-2?api_key=almost-no-requests-per-day'
...
X-ApiaxleProxy-Qpd-Left: 0

curl -v 'service-2?api_key=almost-no-requests-per-day'

...
HTTP/1.1 429 Too Many Requests

{
  "meta": {
    "status_code": 429,
    "version": 1
  },
  "results": {
    "error": {
      "message": "Queries per day exceeded: Queries exceeded (1 allowed).",
      "type": "QpdExceededError"
    }
  }
}
```

Where to go from here

Now that you have this infrastructure setup, all that is left is to integrate with your user management system to manage these APIs and access to them. ApiAxle is very flexible and will even do some statistics analytics for you, which we'll cover in a future post. As a sneak peak you can navigate to **<http://apiaxle-api/#/apis/service-2>** (**<http://apiaxle-api/#/apis/service-2>**).

Although we used docker in this post to demonstrate basic flows, please note that we don't use these docker images in production. For production setup that is battle-tested please refer to our **chef cookbook for ApiAxle** (<https://github.com/mapzen/chef-apiaxe>).

Why we chose ApiAxle over other API management options

When should you deploy something in-stack versus using a hosted 3rd party service for API management? Flexibility and control were our primary criteria, so we were quite pleased when we discovered **ApiAxle** (<https://github.com/apiaxe/apiaxe>) on via **stackoverflow** (<http://stackoverflow.com/questions/533701/is-there-a-free-api-management-system-e-g-a-mashery-alternative>).

As your organization decides how to manage their APIs, you should investigate all the options, including hosted 3rd party services such as **3scale** (<https://www.3scale.net>) and the new **Amazon API Gateway service** (<https://aws.amazon.com/api-gateway/>).

Running your own system comes with one minor disadvantage compared to third-party hosted services – it does not run on the edge. In our case, we don't need to rate-limit our users for responses we've already computed and are served from our edge caches. However, we still measure the usage so we can get a clear picture of what is happening on our APIs.

One thing that should be called out about ApiAxle is that it doesn't appear to be actively maintained, though we've had great success with it in the past year and things have been stable. We've made a couple of adjustments and are in the process of preparing changes to be merged upstream. One is for stats batching and XHR headers configurations. Analytics is the biggest area where ApiAxle could be improved. We plan on building more functionality into the product over the next few months, which will be the topic of a upcoming **Engineering at Mapzen** (<https://mapzen.com/tag/engineering>) blog post.

Conclusion

We've successfully used ApiAxle for over a year and haven't had any problems to speak of. It's stable and gives you great flexibility to manage and craft your services as you choose. ApiAxle focuses on solving a specific problem well and leaves it up to you to implement other requirements like user management.

Let me know (<http://twitter.com/baldur>) if you have any questions on the demo, or on our use of ApiAxle!

· 29 March 2016 ·



Baldur Gudbjornsson

Former VP of Engineering, with special focus on devops, infrastructure, community and api management.

© 2017 Mapzen

AAG Mapathon

osm (/tag/osm) **targeted-editing** (/tag/targeted-editing)

In San Francisco for the AAG (Association of American Geographers) annual meeting? Come by for the AAG's **very first mapathon** (<http://aagmapathon.org/>) and learn how to create and edit data in OpenStreetMap! **There will be keynote sessions and workshops starting at 11:40 each day.** (<http://aagmapathon.org/schedule.html>)

March 30th – April 1st, 2016, San Francisco, CA

AAG Mapathon

Join us for a three day Mapathon and contribute to OpenStreetMap for humanitarian efforts during the [AAG Conference](#). Organized by [MapGive](#) (an initiative of the U.S. Department of State's Humanitarian Information Unit), [George Mason University](#), and [friends](#). Registration to the [AAG Meeting](#) is required to attend the AAG Mapathon. Don't forget to bring your laptop to join the mapping!

[Get Started Mapping](#)

Map designed by [Eric Fisher](#) ©Mapbox ©OpenStreetMap



**Secondary Cities and
Urban Resilience**



**Disaster
Preparedness and
Response**



**Health and Infectious
Disease**

[Get Started Mapping](#)

[Learn More](#)

[Metrics](#)

Screenshot via **AAG Mapathon** (<http://aagmapathon.org/>)

You can get a head start by **checking out Indy's series of posts on Targeted Editing** (<https://mapzen.com/tag/targeted-editing>). **Streets** (<https://mapzen.com/blog/targeted-editing-no-name-roads>)! **Hospitals** (<https://mapzen.com/blog/targeted-editing-hospital-polygons>)! **Schools** (<https://mapzen.com/blog/targeted-editing-school-polygons>)!

On **Thursday at 10 AM** (<http://aagmapathon.org/schedule.html>), Indy Hurt will talk about "Understanding OpenStreetMap: Current-, Historical-, and Meta-Data Analysis" and at 2:20, Drew Dara-Abrams will be giving a **lightning talk in the Disrupt Geo session** (<http://aagmapathon.org/disrupt-geo.html>). Come and say hi! (We also have stickers.)

· 30 March 2016 ·



Indy Hurt

Indy was our resident data scientist lending her geographic expertise to all things "open".



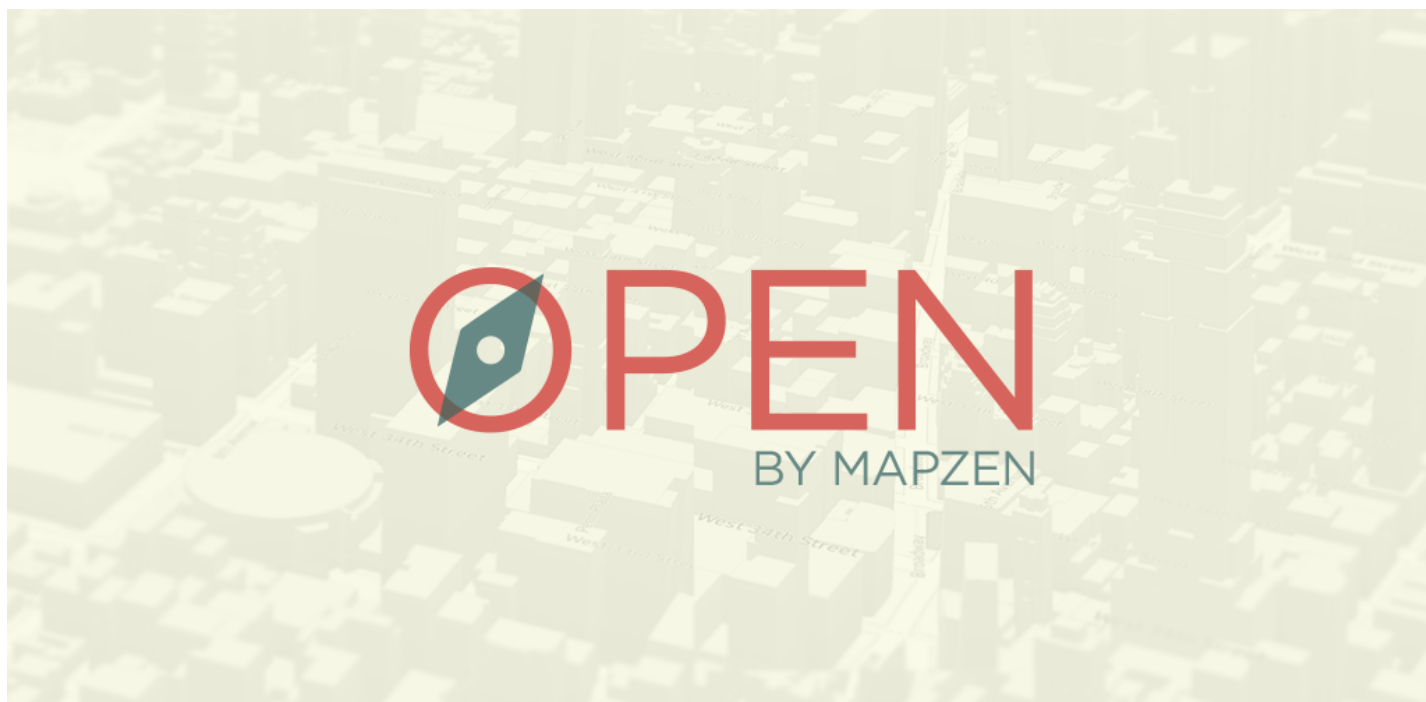
Drew Dara-Abrams

Drew leads Mapzen Mobility products (and aspires to being a flâneur).

© 2017 Mapzen

Open is Closed

erasuremap (/tag/erasuremap) mobile (/tag/mobile) android (/tag/android)



Back in the summer of 2014, when Mapzen was a fraction of the size it is now, **we built an Android app (<https://mapzen.com/blog/we-made-an-app>)** to see if we could cobble together enough open-source software to make a mapping app that anyone could use to get around. We named this app “Open,” and today we’re shutting it down.

This is not one of those “with a heavy heart” kind of things, this is great news. Open proved to us that it was possible to use open data and open-source software to build consumer products, but more importantly, it showed us all of the opportunities for improvement. It inspired us to build **Mapzen Turn-by-Turn (<https://mapzen.com/projects/turn-by-turn>)**, improve **Mapzen Search (<https://mapzen.com/projects/search>)**, and to build **numerous mobile components (<https://mapzen.com/tag/mobile>)**. It allowed us to build on top of our services, just like our developers, and helped shape how we think about **documentation (<https://mapzen.com/documentation/>)** and project structure. It helped us get to where we are today, but it is not our future. We feel that rather than supporting it as an afterthought, it’s better to shut it down completely.

For all the dedicated users of Open, rest assured we have something better for you. Our latest Android efforts are focused on **Eraser Map (<https://mapzen.com/blog/erasermap-beta>)**, which has even more functionality than Open, plus the added benefit of privacy. Eraser Map is in private beta now and we welcome everyone, especially all the users of Open, to **sign up here (<https://www.erasermap.com/>)**.

· 31 March 2016 ·



Mike Cunningham

I'm Street View famous in two cities and I do product @mapzen.

© 2017 Mapzen

Targeted Editing – Campus Mapping

osm (/tag/osm) **targeted-editing** (/tag/targeted-editing)

Maps are current as of Oct 2016.

So you want to do some micro mapping? Let's get started with a campus!

Welcome to our **Targeted Editing** series where today you can explain how the universe is expanding or make OpenStreetMap data outstanding! (Both take about the same amount of time.)

Back in **December** (<https://mapzen.com/blog/targeted-editing-school-polygons>), we published one of our first Targeted Editing posts encouraging editors to add polygons to represent school grounds where the school was currently only represented as a point. See them highlighted in the map below.



Leaflet | Geocoding by Mapzen

(This map is interactive! Open full screen ↗ (<https://mapzen-data.github.io/targeted-editing/te-school-polygons/map/index.html?schools>))

Since that post, there have been a few additional mapathons encouraging editors to contribute schools. The Quarterly Projects for the **UK**

(http://wiki.openstreetmap.org/wiki/UK_Quarterly_Projects) and **Belgium**

(http://wiki.openstreetmap.org/wiki/WikiProject_Belgium/BE_Quarterly_Projects) are

successful examples along with **International Women's Day**

(<https://openstreetmap.us/2016/02/womens-mapathon/>) mapathons held in February.

Looking through just a small subset of cities from our Metro Extracts used for **Targeted Editing**

(<https://mapzen.com/tag/targeted-editing>) posts, 6,591 polygons with the `amenity=school` tag have been added or modified!

Region	City	Total School Polygons	New Since December	% Increase
North America				

Region	City	Total School Polygons	New Since December	% Increase
	Atlanta	474	35	8%
	Boston	351	60	21%
	DC	1,811	97	6%
	Detroit	942	108	13%
	Houston	641	83	15%
	Indianapolis	101	6	6%
	Jacksonville	103	16	18%
	Los Angeles	2,234	227	11%
	Mexico City	663	72	12%
	Miami	630	14	2%
	Minneapolis St Paul	473	13	3%
	New Orleans	131	18	16%
	New York City	1,409	86	7%
	Phoenix	681	92	16%
	San Diego	309	9	3%
	San Francisco Bay Area	2,196	150	7%
	Tampa	317	23	8%
	Vancouver	123	12	11%
South America				
	São Paulo	671	137	26%
Asia				
	Beijing	260	30	13%

Region	City	Total School Polygons	New Since December	% Increase
	Bengaluru	593	89	18%
	Hong Kong	1,593	199	14%
	Moscow	6,804	854	14%
	Mumbai	234	9	4%
	New Delhi	900	16	2%
	Seoul	1,438	171	13%
	Singapore	643	56	10%
	Tokyo	6,988	467	7%
Oceania				
	Auckland	639	236	59%
	Melbourne	1,744	117	7%
	Sydney	975	135	16%
Europe				
	Berlin	2,793	647	30%
	Dublin	610	92	18%
	London	6,024	1825	43%
	Paris	3,932	270	7%
	Stockholm	1,683	120	8%
Totals				
		52,113	6,591	

***SQL queries (https://github.com/mapzen-data/targeted-editing/blob/gh-pages/queries/new_school_sql.sql)** for those interested in generating stats for additional cities.

The **amenity=school** (<http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dschoo>) tag is only intended for school grounds when applied to a polygon, but we have found it applied to multiple campus buildings on a single campus on more than a few occasions. While not the intended use of this tag, we still count all of these features as a success for schools!

Since it was one of our most popular posts, we thought we would revisit schools and outline several additional features editors can add. If you are brand new to editing, you will find lots of editing resources listed at the bottom of this post and all **posts we have published in this series** (<https://mapzen.com/tag/targeted-editing>).

One important note: You may find a campus map online with lots of details you would like to add to OpenStreetMap. Before you start digitizing from a school's campus map, you **must** obtain permission from the school first, and reference the campus map as a **source** (<http://wiki.openstreetmap.org/wiki/Key:source>) for all of your edits. Most schools will be delighted to have their campus featured on OpenStreetMap, so don't be shy in asking. Extensive indoor mapping **absolutely** requires explicit permission for privacy and safety reasons.

Let's get started!

Start with a polygon that covers the grounds of the ENTIRE campus.

Use the following tags:

- **amenity=school** (<http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dschoo>) if the school is for children age 5 to 18.
- **amenity=university** (<http://wiki.openstreetmap.org/wiki/Tag:amenity%3Duniversity>) for universities.
- **amenity=college** (<http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dcollege>) for colleges.

How do you know which amenity type to use for your school grounds when mapping institutions of higher education? You might get some clues from the name of the school. Many contain the word "College" or "University", although some colleges have "School" in the name, are a part of a larger campus, and do not provide education for children age 5 to 18. Use your best judgement.

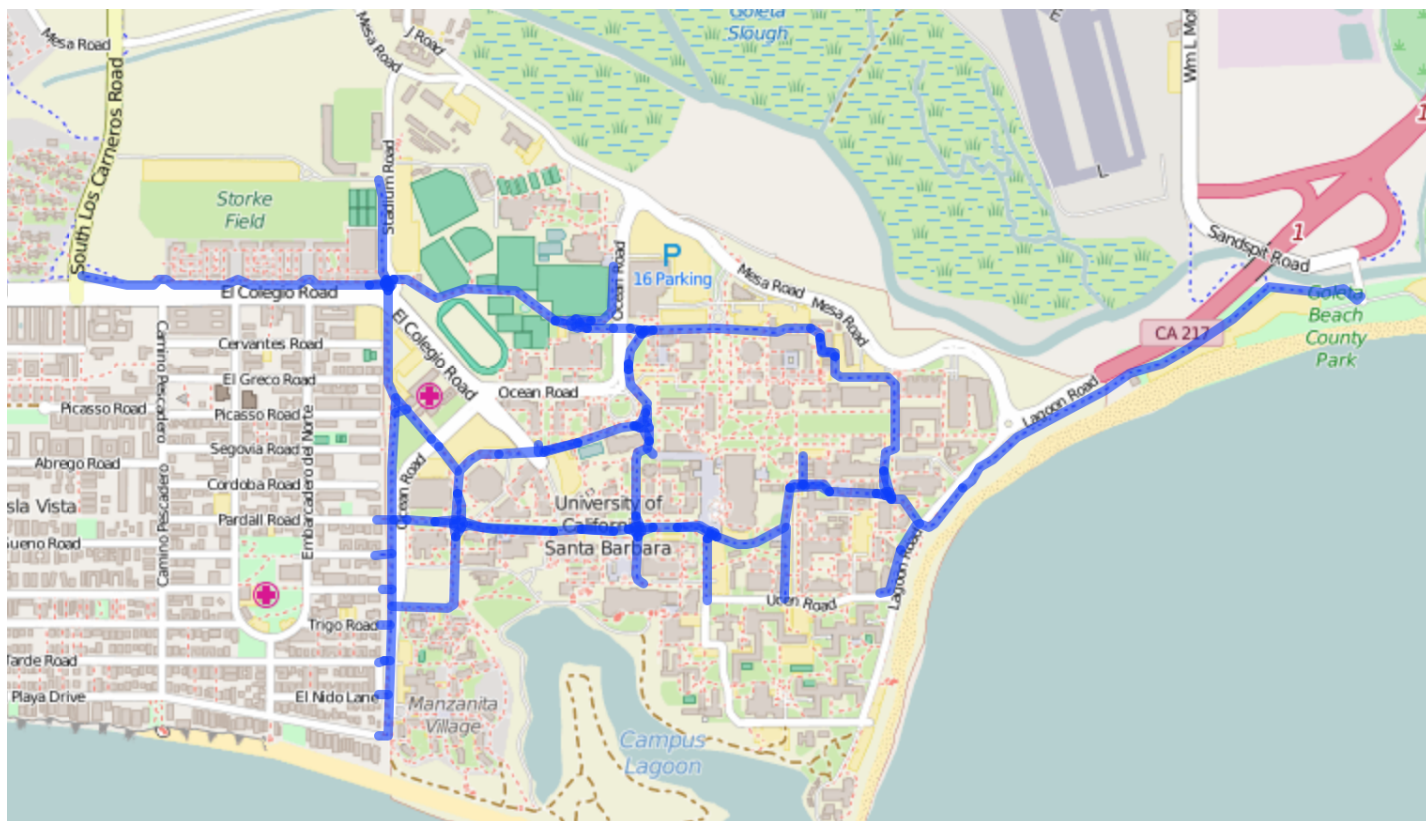
If a campus is comprised of more than one polygon, use a multi polygon **site** (<http://wiki.openstreetmap.org/wiki/Relations/Proposed/Site>) **relation** (<http://wiki.openstreetmap.org/wiki/Relation:multipolygon>), tag each area as **role=outer** , and give the relation the **amenity=school** , **amenity=university** , or **amenity=college** - whichever is most appropriate.

Digitize all walkways and cycling paths on the campus.

Use the following tags:

- **highway=path** (<http://wiki.openstreetmap.org/wiki/Tag:highway%3Dpath>) if the path is multi use but designated for non motorized vehicles.
- **highway=footway** (<http://wiki.openstreetmap.org/wiki/Tag:highway%3Dfootway>) if the path is designated for non motorized vehicles
- **highway=cycleway** (<https://wiki.openstreetmap.org/wiki/Tag:highway%3Dcycleway>) if the path is designated for bicycles only
- **surface** (<https://wiki.openstreetmap.org/wiki/Key:surface>)=* to describe the surface of the path

UCSB has a lot of bike paths on campus. See them highlighted below with a map generated with **Overpass Turbo** (<http://overpass-turbo.eu/s/fjY>).



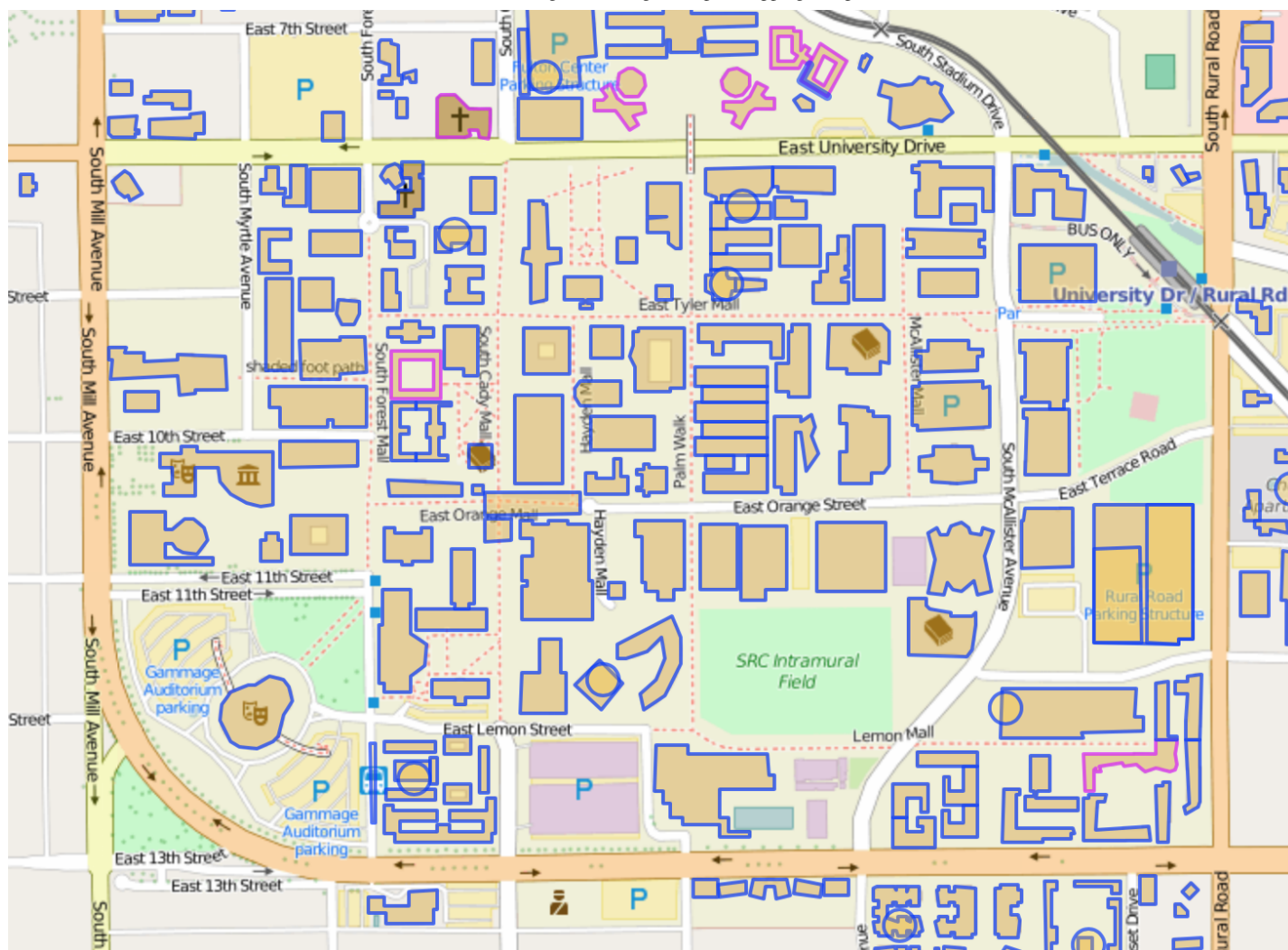
Digitize the buildings as polygons.

Use the following tags:

- **building=school** (<https://wiki.openstreetmap.org/wiki/Tag:building%3Dschooll>)
- **building=university** (<https://wiki.openstreetmap.org/wiki/Tag:building%3Duniversity>)
- **building=college** ()

- **building=dormitory**
(<https://wiki.openstreetmap.org/wiki/Tag:building%3Ddormitory>) or
building=hall_of_residence
- **amenity=library** (<https://wiki.openstreetmap.org/wiki/Tag:amenity%3Dlibrary>) for
libraries together with building=yes
- **height** (<https://wiki.openstreetmap.org/wiki/Key:height>)=* to associate a height of a
building in **METERS** and/or **building:levels**
(<https://wiki.openstreetmap.org/wiki/Key:building:levels>) for the number of floors
above ground.
- use a **multipolygon relation**
(<http://wiki.openstreetmap.org/wiki/Relation:multipolygon>) to digitize a building with
an interior **courtyard**
(http://wiki.openstreetmap.org/wiki/Proposed_features/courtyard).
- **name** (<http://wiki.openstreetmap.org/wiki/Names>)=* (populate with the name of the
building)

All of the buildings on the Arizona State University campus appear to be mapped, and maybe even an *extra* one in the parking lot off East University Drive. Do these buildings have additional useful tags? See the buildings highlighted below with a map also generated by **Overpass Turbo** (<http://overpass-turbo.eu/s/fk0>).



Digitize the entrances of buildings as points.

Use the following tags:

- **entrance=yes** (<http://wiki.openstreetmap.org/wiki/Key:entrance>)
- **entrance=main** (<http://wiki.openstreetmap.org/wiki/Key:entrance>)
- **entrance=emergency, etc** (<http://wiki.openstreetmap.org/wiki/Key:entrance>)
- Ensure that the walking paths connect to the building entrances. This connectivity can be used for routing on campus. Researchers interested in evaluation models will find this extremely valuable.

When multiple departments exist in a building, digitize the departments as nodes.

Use the following tags:

- **office=yes** (<http://wiki.openstreetmap.org/wiki/Key:office>)
- **name** (<http://wiki.openstreetmap.org/wiki/Names>)=* (populate with the name of the department)

- **level** (<http://wiki.openstreetmap.org/wiki/Key:level>)=* (to indicate what floor the department office is on - not to be confused with **building:levels** (<https://wiki.openstreetmap.org/wiki/Key:building:levels>))
- **all appropriate address fields** (<http://wiki.openstreetmap.org/wiki/Addresses>)
- **phone** (<http://wiki.openstreetmap.org/wiki/Key:phone>)=*
- **website** (<http://wiki.openstreetmap.org/wiki/Key:website>)=*

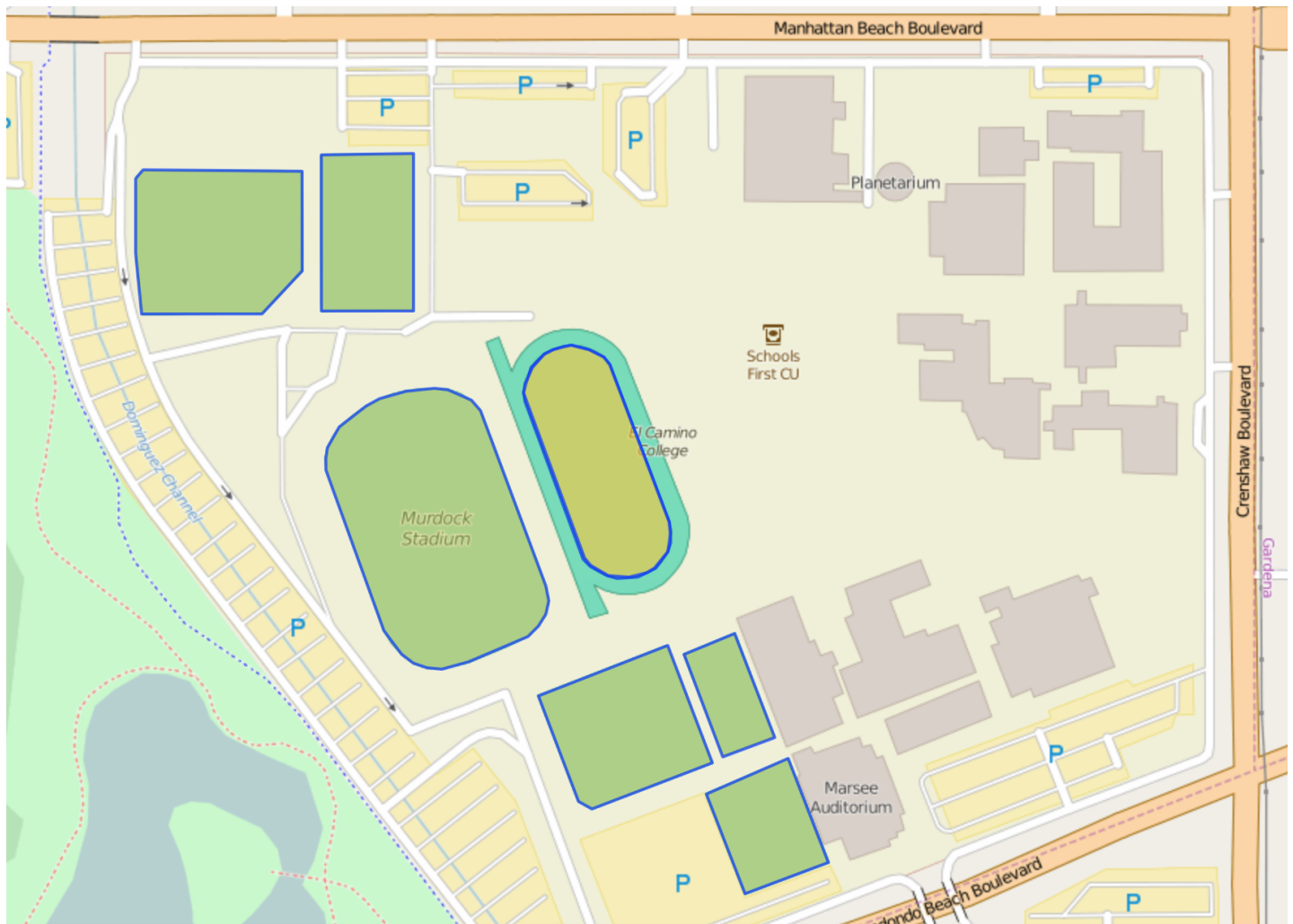
Mapping departments on campuses is not as wide spread as one might think, but they are one of the most important campus features. A few existing keys include `faculty=*` and `department=*` each representing a little over 400 features, each. Since all departments have offices, the `office` key provides an alternative mapping approach that can be used in conjunction with the `name` key to provide departments with proper names that search and label engines can use. It is an approach similar to how one might add individual shops to a shopping mall or an airport terminal. Of course, the truly ambitious editor can add all interior spaces of a public building as polygons to get the chorus of **indoor mapping** (http://wiki.openstreetmap.org/wiki/Indoor_Mapping) enthusiasts singing!

Digitize the playing fields & general recreation grounds as polygons.

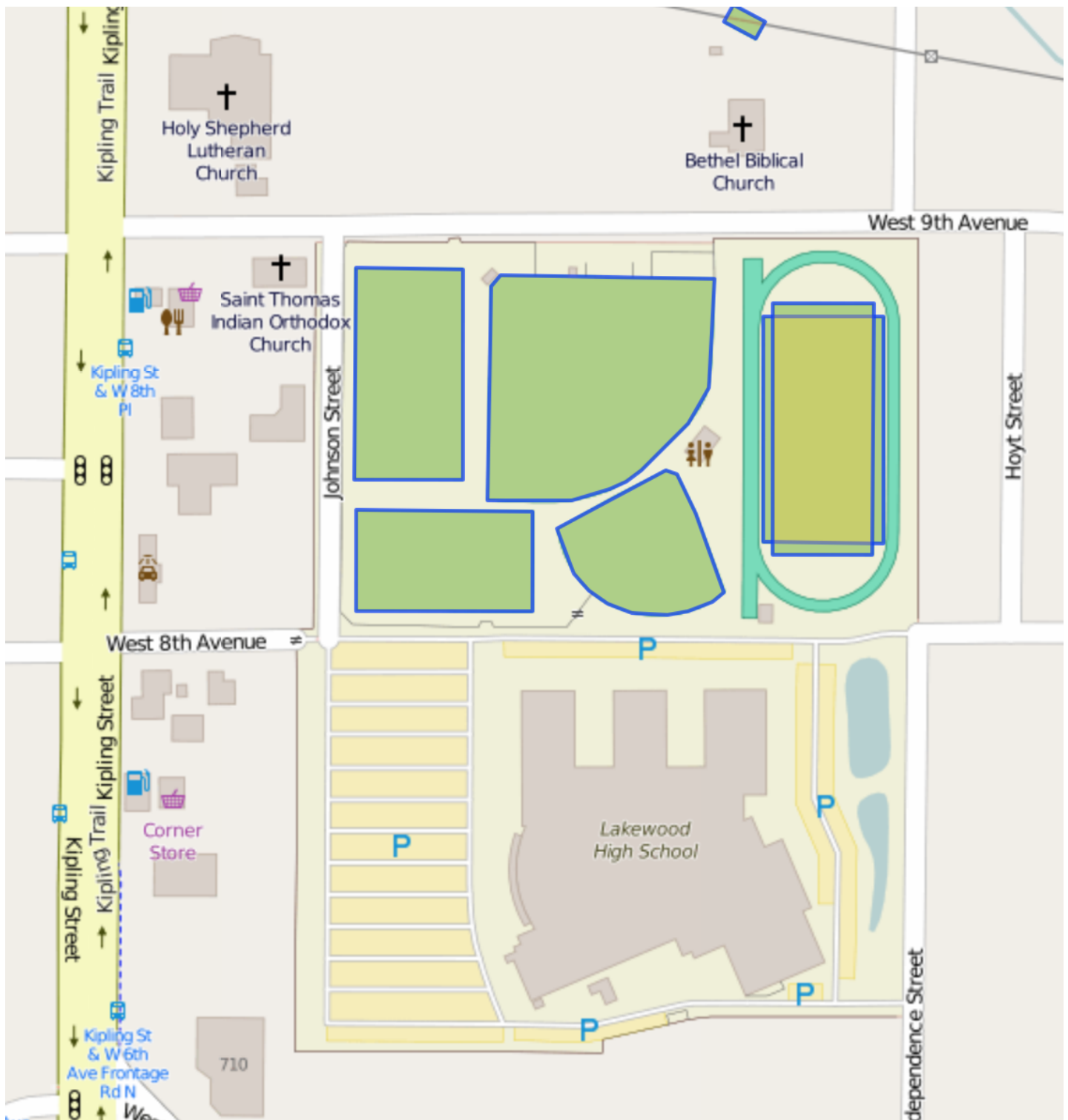
Use the following tags:

- **leisure=pitch** (<http://wiki.openstreetmap.org/wiki/Tag:leisure%3Dpitch>)
- **sport=baseball** (<http://wiki.openstreetmap.org/wiki/Key:sport>), **sport=basketball** (<http://wiki.openstreetmap.org/wiki/Key:sport>), **sport=tennis** (<http://wiki.openstreetmap.org/wiki/Key:sport>), etc.
- **landuse=recreation_ground** (http://wiki.openstreetmap.org/wiki/Tag:landuse%3Drecreation_ground)

Here is a link to an example of pitches editors have added to El Camino College in Los Angeles **Overpass Turbo** (<http://overpass-turbo.eu/s/flf>).



Here is another example for Lakewood High School near Denver via **Overpass Turbo** (<http://overpass-turbo.eu/s/flp>)



Digitize the parking areas as polygons.

Use the following tags:

- **amenity=parking** (<http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dparking>)

Digitize the parking aisles as lines.

Use the following tags:

- **highway=service** (<http://wiki.openstreetmap.org/wiki/Tag:highway%3Dservice>)
- **service=parking_aisle**
(http://wiki.openstreetmap.org/wiki/Tag:service%3Dparking_aisle)
- **oneway=yes** (<http://wiki.openstreetmap.org/wiki/Key:oneway>) (where appropriate).
The direction will be the direction in which the way was digitized. If the direction is incorrect, use the “reverse way” tool to reverse the direction of the way to match the flow of oneway traffic. This may also require adding additional nodes to the way to ensure that tags are applied to the appropriate segments.

If you are feeling extra ambitious, you can even digitize the individual parking spaces! Mapbox engineer and super editor Arun Ganesh provides a great explanatory gif in his **OpenStreetMap Diary** (<http://www.openstreetmap.org/user/PlaneMad/diary/38093>) to make the process of digitizing parking spaces quick and easy with the JOSM editor.

Digitize any designated small bicycle parking areas as points and large bicycle parking areas as polygons.

Use the following tags:

- **amenity=bicycle_parking**
(http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dbicycle_parking)

Digitize lawns and other small grassy areas as polygons.

Use the following tags:

- **landuse=grass** (<http://wiki.openstreetmap.org/wiki/Tag:landuse%3Dgrass>)

Digitize the trees as nodes.

Use the following tags:

- **natural=tree** (<http://wiki.openstreetmap.org/wiki/Tag:natural%3Dtree>)
- **height** (<http://wiki.openstreetmap.org/wiki/Key:height>)=* in meters
- **taxon** (<http://wiki.openstreetmap.org/wiki/Key:taxon>)=* for the family if know - see other tag combinations

Here's an example of trees digitized around the Marymount School of New York via **Overpass Turbo** (<http://overpass-turbo.eu/s/fnl>)



Digitize ATMs (automatic teller machines) as nodes

Use the following tags:

- **amenity=atm** (<http://wiki.openstreetmap.org/wiki/Tag:amenity%3Datm>)
- **operator** (<http://wiki.openstreetmap.org/wiki/Key:operator>)=* (the name of the bank that operates the ATM)

Digitize restaurants in a food court as nodes.

Use the following tags:

- **amenity=restaurant**
(<http://wiki.openstreetmap.org/wiki/Tag:amenity%3Drestaurant>)
- **name** (<http://wiki.openstreetmap.org/wiki/Names>)=*
- **cuisine** (<http://wiki.openstreetmap.org/wiki/Key:cuisine>)=*

- **opening_hours** (http://wiki.openstreetmap.org/wiki/Key:opening_hours)=*
 - See tips for formatting hours in our **fitness** (<https://mapzen.com/blog/new-years-resolutions-fitness>) post or check out **YoHours** (<http://github.pavie.info/yohours/>) for an intuitive interface that can help format hours for you.

No doubt, there are many other things you could map like information kiosks, phone booths, map directory points, sidewalks, even hedges that double as fences! How do you decide? The best bet is to visit a campus, walk around, and catalog the permanent immovable things you would like to map. If you need some help with mapping tools for OpenStreetMap, read on:

Getting Started with OpenStreetMap

Need instructions on how to edit with iD? Here are some links to outstanding tutorials from LearnOSM, the OpenStreetMap wiki, and the United States Department of State's Humanitarian Information Unit:

- **LearnOSM** (<http://learnosm.org/en/>)
- **OSM Beginners' Guide** (http://wiki.openstreetmap.org/wiki/Beginners'_guide)
- **MapGive Learn to Map** (<http://mapgive.state.gov/learn-to-map/>)

Are you a mapping wiz and interested in a more advanced editor? Try out **JOSM** (<https://josm.openstreetmap.de>) with **excellent documentation** (<https://www.mapbox.com/blog/osm-mapping-guide/>) from Mapbox.

Editing with a Mobile Device

With points of interest, I bet you are interested in a mobile app to edit OpenStreetMap while you are on the go.

iOS users can check out **Go Map!!** (<https://appsto.re/us/dafwj.i>) by Bryce Cogswell. This free editor allows you to add features and edit existing features. Check out the **Go Map!!** (http://wiki.openstreetmap.org/wiki/Go_Map!!) wiki page for more details. **Pushpin** (<http://www.pushpinosm.org>) by Fulcrum is another excellent iOS editor for OpenStreetMap points of interest.

Looking for an Android equivalent? **Vespucci** (<https://play.google.com/store/apps/details?id=de.blau.android>) by Marcus Wolschon is a great option. Check out the **Vespucci** (<http://vespucci.io>) home page for documentation and tutorials.

Get outside, increase your local knowledge, and share it with the world through your OpenStreetMap edits!

Check out all the posts in the **Targeted Editing series** (<https://mapzen.com/tag/targeted-editing>):

- **Airports** (<https://mapzen.com/blog/targeted-editing-airport-polygons>)
- **Banks** (<https://mapzen.com/blog/new-years-resolutions-money/>)
- **Building Names** (<https://mapzen.com/blog/targeted-editing-name-that-building>)
- **Campus Mapping** (<https://mapzen.com/blog/targeted-editing-campus-mapping/>)
- **Cycleways** (<https://mapzen.com/blog/targeted-editing-cycleways/>)
- **Fitness** (<https://mapzen.com/blog/new-years-resolutions-fitness>)
- **Groceries** (<https://mapzen.com/blog/new-years-resolutions-groceries>)
- **Hospitals** (<https://mapzen.com/blog/targeted-editing-hospital-polygons>)
- **Schools** (<https://mapzen.com/blog/targeted-editing-school-polygons>)
- **Stadiums & Parking** (<https://mapzen.com/blog/targeted-editing-tailgate-mania>)
- **Street Names** (<https://mapzen.com/blog/targeted-editing-no-name-roads>)
- **Transit Colours** (<https://mapzen.com/blog/targeted-editing-transit-colours>)
- **Travel & Lodging** (<https://mapzen.com/blog/new-years-resolutions-travel>)

· 31 March 2016 ·



Indy Hurt

Indy was our resident data scientist lending her geographic expertise to all things "open".

© 2017 Mapzen

Who's on First? It's Mapzen Search

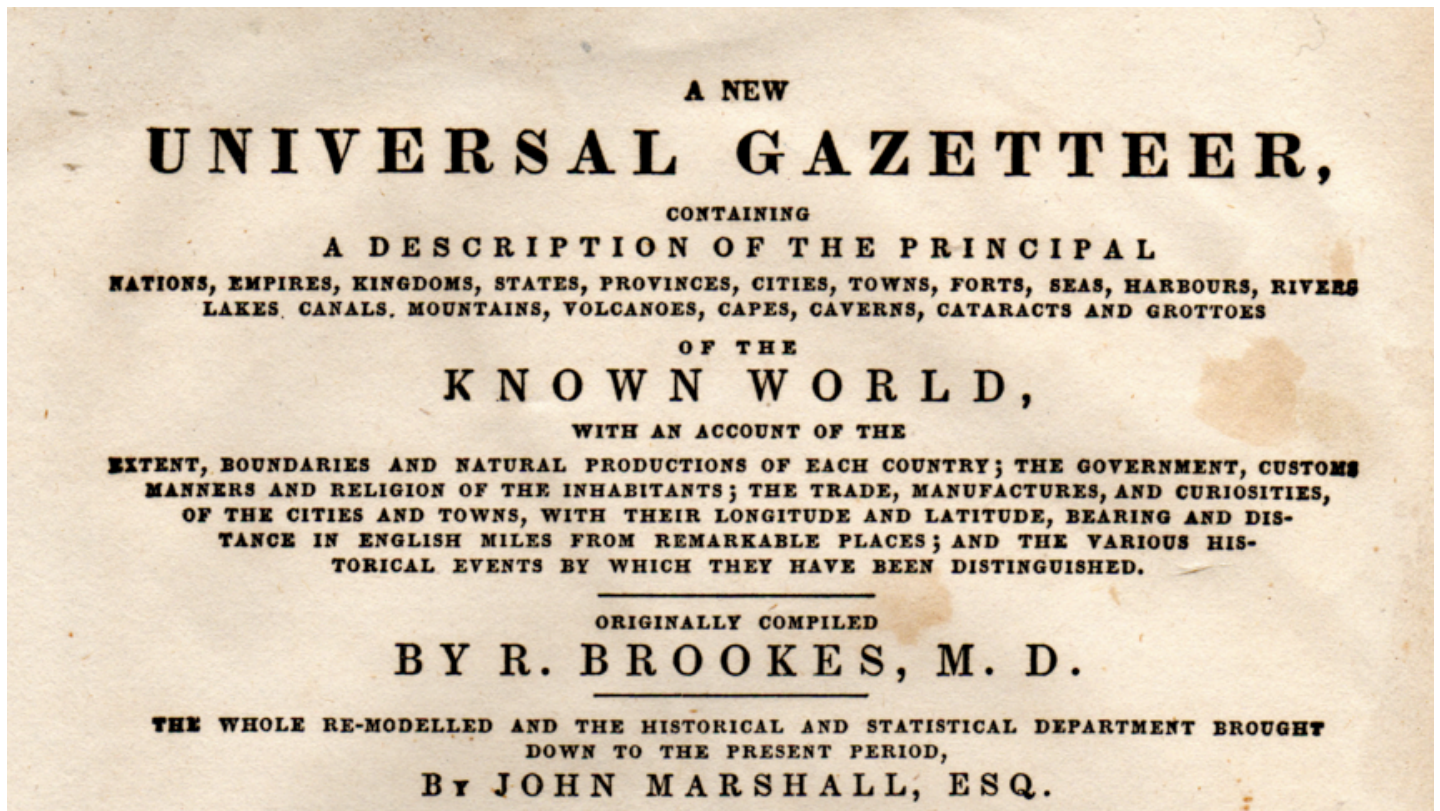
search (/tag/search) **data** (/tag/data) **whosonfirst** (/tag/whosonfirst)

When we launched the first version of the Mapzen Search API in October 2015, we always planned to keep continually improving the service under the hood. We're about to make our biggest change yet, adding the data service **Who's on First** (<https://whosonfirst.mapzen.com/>) to Mapzen Search. *Who's on First* is a gazetteer - a directory of places, linking their names to their locations - one that's built to enable all kinds of forward-looking utilities and features. As with all data powering Mapzen Search, *Who's on First* is 100% open data. And while this change is important in situating Mapzen Search for the future, users won't need to make any changes to start seeing its benefits.

Switching to *Who's on First* as our core gazetteer means there will be some other implications for our underlying data. We'll be removing our current gazetteer, **Quattroshapes** (<http://quattroshapes.com/>) (though it will live on through its data's inclusion in *Who's on First*). Now, when you do a search or geocode, you'll be getting results by default from *OpenStreetMap* (addresses, streets, places of interest), *OpenAddresses* (addresses), Geonames (venues, areas of interest), and *Who's on First* (named areas). Existing users don't need to worry: you'll be able to continue making the exact same API calls you're making today; you'll be getting the same data from *Who's on First*.

For self-hosted users of Pelias, we're upgrading our data importers to use *Who's on First*, but they'll still work if you rely on Quattroshapes for your services as well.

You might be wondering what part something that sounds like a 19th century newspaper plays in a modern geocoder?



Title page of the Brook's Gazetteer, via GEDCOM Index
 (<http://gedcomindex.com/Reference/brookes.html>)

Gazetteers are indexes that connect the names of places to their geographic locations. And they're incredibly useful for connecting named places (and information about those named places) to locations. In *Mapzen Search* the gazetteer plays a special role, in that we use it to find "named places" (countries, regions, continents, cities, and neighborhoods), but also use it to find what places fall inside of *those* places, so that when we get a data source like OpenAddresses or OpenStreetMap that's not guaranteed to have the city or state for a particular venue or address, we can add that information in automatically.

Geonames is perhaps the most used open data gazetteer in the world, originating **125 years ago** (<http://geonames.usgs.gov/anniversary/index.html>) as part of the United States' Board on Geographic Names. It forms a remarkably comprehensive directory of place names, place types, some name translations, and the point on the map that represents that place. Lots of folks use it.

But it's not the only open gazetteer out there.

There's *Where on Earth*, Yahoo!'s gazetteer (also known as *Yahoo! Geoplanet*, which took the time to translate most worldwide placenames into seven different languages and find common nicknames or colloquial names. In addition, it took the importance of situating each place in a

global hierarchy (e.g.

locality:Boston>county:Suffolk>region:Massachusetts>country:United States of America>continent:North America>planet:Earth), and ensuring it provided the records for those places, not just their names. For several years Yahoo! released this dataset under a Creative Commons license.

There's also **Natural Earth** (<http://www.naturalearthdata.com/>), a copyright-free global dataset from Nathaniel Vaughn Kelso and Tom Patterson. It represents most places as polygons, which is a far better way to represent things with borders points, like many of these other gazetteers have. This was further evolved with **Quattroshapes** (<http://quattroshapes.com/>), (which built on **Alphashapes** (<https://code.flickr.net/2008/10/30/the-shape-of-alpha/>) from Flickr and **Betashapes** (<https://github.com/simplegeo/betashapes/commit/4c143cc3367d54a9a1589775ccec30e13696abf6>) from SimpleGeo in addition to Natural Earth), created by Vaughn Kelso and David Blackman at Foursquare. Foursquare wanted a gazetteer that could be used to reverse geocode places worldwide, to find out where their users where, but also to find out which cities and neighborhoods places were located in.

All of these form the core of *Who's on First*, aggregating all their best properties.

From *Natural Earth* and *Quattroshapes*, *Who's on First* has polygons to represent (most) places. It means we don't just find the named places, we can find what places are within *those* places. From *Where on Earth* we have the rich global place hierarchy, so we can represent the complex political structures of governance. This lets us have places like the **United Nations headquarters** (<http://www.openstreetmap.org/way/48869328>) be **legally run like its own country** (<https://whosonfirst.mapzen.com/spelunker/id/102312305/>) while falling within the territorial boundaries of **New York City** (<https://whosonfirst.mapzen.com/spelunker/id/85977539/>). Or representing places with many names, **with their many names** (<https://whosonfirst.mapzen.com/spelunker/id/102026327/>), in many languages.

The Future

What excites us most about *Who's on First* is what it portends for the future of *Mapzen Search*. It's the open data platform we'll need to make a world class geocoder that works around the world.

For the first time, we can use a polygon-based gazetteer that's getting constant editorial attention, so as the world changes, we can keep it up-to-date with the times. New places, updated populations, even **temporary places**

(<https://whosonfirst.mapzen.com/spelunker/id/420561633/>) will make their way to Mapzen Search through *Who's on First*.

Who's on First will also help us bring multiple languages and common nicknames into Mapzen Search. With multiple languages for nearly every country, region, and major city, you'll be searching for "The Big Apple" or "Nueva York" in no time.

And *Who's on First* is the new home of over 8 million venues from **SimpleGeo** (<https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces>), which we hope to make available soon for more point-of-interest searches based on open data.

Who's on First is exactly what's next for *Mapzen Search*. And it's going to make finding places around the world a lot easier.

· 05 April 2016 ·



David Riordan

Former product manager for Mapzen Search. Historical mapping, open tools, open access, and open data.

© 2017 Mapzen

Yes No Fix

whosonfirst (/tag/whosonfirst)

Yes No Fix

tl;dr

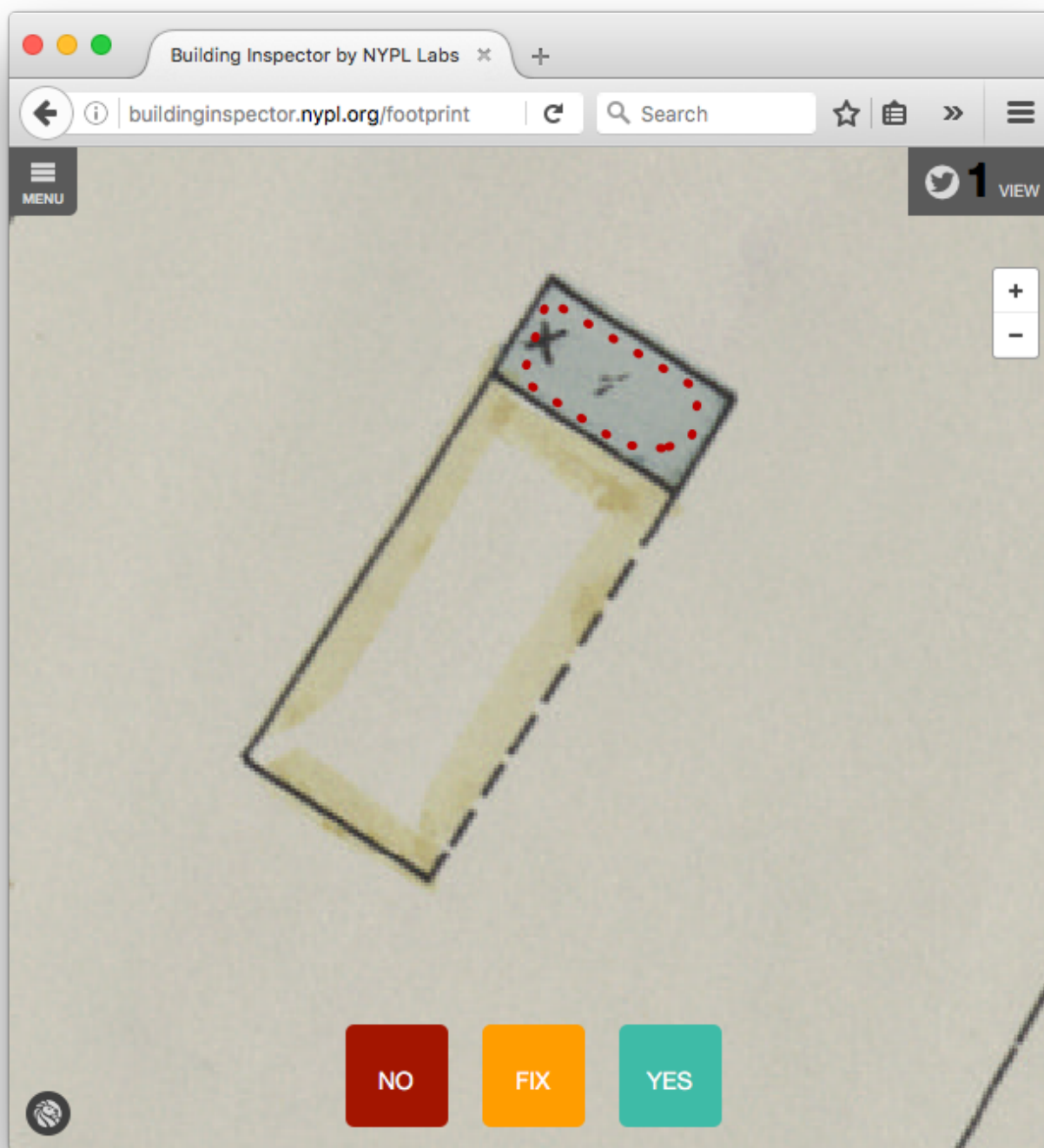
Opinions and fact-checking. About stuff. As CSV documents. From any webpage. Or at least **Who's On First Spelunker** (<https://whosonfirst.mapzen.com/spelunker/>) webpages. With code (<https://github.com/whosonfirst/js-mapzen-whosonfirst-yesnofix>).

A short history

In the list of *Really Good Things on the Internet These Days* I still think the work done by the **Government Digital Service** (<https://gds.blog.gov.uk/>) in the United Kingdom to rebuild the **gov.uk** (<https://gov.uk>) website is at, or near, the top. A close second would be the New York Public Library's (NYPL) **Building Inspector** (<http://buildinginspector.nypl.org/>) project.

The Building Inspector project began after the NYPL developed **a suite of computer vision tools for extracting the building footprints** (<https://github.com/nypl-spacetime/map-vectorizer>) from their extensive maps collection. The results were remarkably good for an automated process but still not perfect. Occasionally the software would get things wrong but sometimes it would return results that were maybe not wrong but not entirely correct either. To deal with these inconsistencies the NYPL created the **Building Inspector website** (<http://buildinginspector.nypl.org/footprint/>) and asked the public to help out by asserting whether a footprint computed by their software was correct (yes), incorrect (no) or in need of some fixing (fix).

It looks like this:



The **project** (<https://github.com/nypl-spacetime/building-inspector>) has since evolved to allow contributors to vet more than just building footprints and the NYPL recently announced **that people have contributed more than 1.5 million tasks** (https://twitter.com/nypl_labs/status/716999124319670272) since launching. That original concept to atomize the problem (individual building footprints) and then ask the public for simple observations (yes, no, fix) remains a stroke of genius. So we thought we'd try copying it.

How does it work

Yes No Fix is a single Javascript library with a pair of public API methods. Each method takes an arbitrary data structure as its input and renders it as a nested HTML table where each value (at the end of a nesting) has interactive controls to allow a viewer to assert an opinion (yes, no or fix) about that value. The second method will also append the rendered table to an existing DOM element in the webpage it was called from.

Here's an example using the `names` section for **the city of San Francisco** (<https://whosonfirst.mapzen.com/spelunker/id/85922583/>) from the Who's On First Spelunker (<https://whosonfirst.mapzen.com/spelunker/>). The raw data looks like this:

```
"properties": {
  "name:chi_x_preferred": [
    "\u65e7\u91d1\u5c71"
  ],
  "name:chi_x_variant": [
    "\u820a\u91d1\u5c71"
  ],
  "name:eng_x_colloquial": [
    "City by the Bay",
    "City of the Golden Gate",
    "Fog City",
    "Fog Cty",
    "Frisco",
    "Golden City",
    "S Fran",
    "S. Fran",
    "San Fran",
    "The City",
    "S.F.",
    "Bay Area",
    "S.F. Bay Area",
    "The City by the Bay",
    "Baghdad by the Bay",
    "The Paris of the West",
    "Ess Eff",
    "SFC",
    "San Francisco City"
  ],
  // and so on
```

And the rendered version looks like this:

name	
chi_x_preferred	旧金山
chi_x_variant	舊金山
eng_x_colloquial	City by the Bay
	City of the Golden Gate
	Fog City
	Fog Cty
	Frisco
	Golden City
	S Fran
	S. Fran
	San Fran
	The City
	S.F.
	Bay Area
	S.F. Bay Area
	The City by the Bay
	Baghdad by the Bay
	The Paris of the West
	Ess Eff
	SFC
	San Francisco City
eng_x_preferred	San Francisco
eng_x_unknown	SF

When you mouse over a value - in this case the English colloquial name of **The City** - you'll see an edit control.

name

chi_x_preferred	旧金山
chi_x_variant	舊金山
eng_x_colloquial	City by the Bay
	City of the Golden Gate
	Fog City
	Fog Cty
	Frisco
	Golden City
	S Fran
	S. Fran
	San Fran
	 The City
	S.F.
	Bay Area
	S.F. Bay Area
	The City by the Bay
	Baghdad by the Bay
	The Paris of the West
	Ess Eff
	SFC
	San Francisco City

If you click on it then a series of controls (yes, no and fix) will appear next to that value. Like this:

Yes

The first is **yes** which means that you agree with the value (and its parent nesting).

name

chi_x_preferred

旧金山

chi_x_variant

舊金山

eng_x_colloquial

City by the Bay

City of the Golden Gate

Fog City

Fog Cty

Frisco

Golden City

S Fran

S. Fran

San Fran



The City

yes

no

fix

cancel

?

S.F.

Bay Area

S.F. Bay Area

The City by the Bay

Baghdad by the Bay

The Paris of the West

Ess Eff

SFC

San Francisco City

No

The second is **no**. **No means no.** (https://en.wikipedia.org/wiki/No_means_no) San Francisco is not called *Frisco*.

name

chi_x_preferred

旧金山

chi_x_variant

舊金山

eng_x_colloquial

City by the Bay

City of the Golden Gate

Fog City

Fog Cty

**Frisco**

yes

no

fix

cancel

?

Golden City

S Fran

S. Fran

San Fran

The City

S.F.

Bay Area

S.F. Bay Area

The City by the Bay

Baghdad by the Bay

The Paris of the West

Ess Eff

SFC

San Francisco City

[Ed. Some of us think *Frisco* **is** (<https://www.thrillist.com/entertainment/san-francisco/san-francisco-nickname-frisco-sf>) actually **OK** (<http://www.buzzfeed.com/burritojustice/frisco-24wct>). Which kind of proves the point of Yes No Fix. *San Fran* on the other hand...]

name

chi_x_preferred

旧金山

chi_x_variant

舊金山

eng_x_colloquial

City by the Bay

City of the Golden Gate

Fog City

Fog Cty

Frisco

Golden City

S Fran

S. Fran



San Fran

yes

no

fix

cancel

?

The City

S.F.

Bay Area

S.F. Bay Area

The City by the Bay

Baghdad by the Bay

The Paris of the West

Ess Eff

SFC

San Francisco City

Fix

The third value is **fix** which means broadly *this is weird data*. If that seems a little vague and ambiguous that is because it's meant to be. "Fix" is a shorthand for things that are sort of correct and but still incorrect or vice versa. Life is complicated that way.

name

chi_x_preferred

旧金山

chi_x_variant

舊金山

eng_x_colloquial

City by the Bay

City of the Golden Gate

Fog City

Fog Cty

Frisco

Golden City

S Fran

S. Fran

San Fran

The City

S.F.

Bay Area

S.F. Bay Area

The City by the Bay

Baghdad by the Bay

The Paris of the West



Ess Eff

yes

no

fix

cancel

?

SFC

San Francisco City

Locked

Sometimes a value might be “locked” or “excluded” which means that it is not possible to make a `yesnofix` style assertion about it. The reasons why something might be excluded are defined by individual applications. We’ll explain how that’s done, below. In this example the `edtf:inception` and `edtf:cessation` dates are locked because they already have a default value of “unknown” so there’s not a lot of use in collecting opinions about them.

Properties — *some notes about sources and names*

[view raw](#)
[show report](#)

edtf

cessation

uuuuu

inception

 uuuuu

geom

area

0.061408

bbox

-123.173825,37.63983,-122.28178,37.929824

latitude

37.759715

longitude

-122.693976

Reports

In the screenshot above there is a `show report` button. When clicked it will display three more elements: A comma-separated value (CSV) rendering of all the assertions that have been made so far and another button for submitting the report (and a button to hide everything).

Properties — *some notes about sources and names*

[view raw](#)
[hide report](#)
[submit report](#)

```
path,value,assertion,date
```

```
name.eng_x_colloquial#10,S.F.,1,2016-04-02T17:55:36.335Z
```

```
name.eng_x_colloquial#9,The City,1,2016-04-02T17:55:50.767Z
```

```
name.eng_x_colloquial#4,Frisco,0,2016-04-02T17:56:17.137Z
```

```
name.eng_x_colloquial#16,Ess Eff,-1,2016-04-02T17:57:38.917Z
```

edtf

cessation

uuuuu

inception

uuuuu

By default, Yes No Fix will “submit” the report to a new browser window because that’s all it knows how to do. Here’s a plain-text version of the report shown in the screenshot above:

```
path,value,assertion,date
name.eng_x_colloquial#10,S.F.,1,2016-04-02T17:55:36.335Z
name.eng_x_colloquial#9,The City,1,2016-04-02T17:55:50.767Z
name.eng_x_colloquial#4,Frisco,0,2016-04-02T17:56:17.137Z
name.eng_x_colloquial#16,Ess Eff,-1,2016-04-02T17:57:38.917Z
```

Reports are formatted as CSV documents, with four columns:

1. `path` represents the nested keys from your data structure (in this case the `properties` dictionary from the underlying **GeoJSON file for San Francisco ()** collapsed in to a string using a `.` notation as a delimiter.
2. `value` is the raw value that someone is commenting on.
3. `assertion` is a (signed) integer; `1` means yes, `0` means no and `-1` means fix.
4. `date` is an ISO-8601 date string indicating when the assertion was made

A couple things to note about paths:

- As of this writing there are still some explicit Who’s On First -isms left in the Yes No Fix code. Specifically the expectation that keys have a colon-separated prefix (for example `name:eng_x_colloquial`) that is parsed and used to group things in to buckets. Keys that don’t have a prefix are automatically grouped in to bucket called `_global_`, so if you had a key simply called `date` it would be encoded as `_global_.date` in the final CSV report.
- Array values in a path are denoted using a `#<OFFSET>` syntax. For example `name.eng_x_colloquial#16` is the 16th element in the `properties['name:eng_x_colloquial']` array.

How to use `yesnofix.js`

First grab a copy of the code from the **js-mapzen-whosonfirst-yesnofix GitHub repository** (<https://github.com/whosonfirst/js-mapzen-whosonfirst-yesnofix>). Then add it to your webpages, like this:

```
<link rel="stylesheet" type="text/css" href="mapzen.whosonfirst.yesnofix.css" />
<script type="text/javascript" src="mapzen.whosonfirst.yesnofix.js"></script>
```

The simplest way to use `yesnofix.js` is to call the `apply` method with a target HTML element and a data structure. This will generate a pretty HTML table complete with Yes No Fix style controls for each value and insert it in to the DOM as a child of the target HTML element you've defined.

```
mapzen.whosonfirst.yesnofix.apply(data, target_el);
```

If you just want to render a data structure but delay or defer adding it to the DOM you can call the `render` method.

```
var pretty = mapzen.whosonfirst.yesnofix.render(data);
```

That's it. By default every element in your data structure will be made Yes No Fix -able.

Customizing things

*Warning: This is the part where things start to get a bit nerdy. Where "a bit nerdy" really means **VERY VERY NERDY**. If you're not in to the nerdy bits you should have enough information to get started and can just **jump to the bottom of the post**.*

One of the things that quickly became apparent integrating `yesnofix.js` with the Who's On First spelunker is that many things needed to be customized. The whole point of a spelunker is to be able to **jump around between documents** (<https://mapzen.com/blog/spelunker-jumping-into-who-s-on-first/>) so at a minimum we would need a way to teach the `yesnofix.js` rendering code to display certain things (like IDs) as links.

Customing things - Values

To do this for values you need to invoke the `set_custom_renderers` method passing "text" as the first argument and a custom function as the second argument. This function will be invoked for each value that the `yesnofix.js` code tries to render.

Your custom function will be invoked with two arguments: `data` which is the actual value in question and `ctx` which is the nested path in dot notation (described above) that contains `data`. Your custom function is expected to either return a function (that itself returns an HTML DOM element) or `null`. If your callback's response is `null` then the code will simply include the raw value as-is.

The `yesnofix.js` code defines some handy helper methods for common tasks (like `render_code` or `render_link`) but in the example below you can see how we are also defining some custom methods, like `render_wof_id`.

```

var possible_wof = [
  'wof.belongsto',
  'wof.parent_id', 'wof.children',
  // as so on...
];

var text_callbacks = {
  'wof.id': mapzen.whosonfirst.yesnofix.render_code,
  // and so on...
};

var text_renderers = function(d, ctx){

  if ((possible_wof.indexOf(ctx) !== -1) && (d > 0)){
    return self.render_wof_id;
  }

  else if (text_callbacks[ctx]){
    return text_callbacks[ctx];
  }

  // and so on...

  else {
    return null;
  }
};

'render_wof_id': function(d, ctx){

  var root = mapzen.whosonfirst.spelunker.abs_root_url();
  var link = root + "id/" + encodeURIComponent(d) + "/";
  var el = mapzen.whosonfirst.yesnofix.render_link(link, d, ctx);

  var text = el.children[0];
  text.setAttribute("data-value", mapzen.whosonfirst.php.htmlspecialchars(d));
  text.setAttribute("class", "props-uoc props-uoc-name props-uoc-name_" + mapzen.whosonfirst.yesnofix.render_wof_id(d));

  return el;
}

mapzen.whosonfirst.yesnofix.set_custom_renderers('text', text_renderers);

```

In this example we are rendering things that are WOF IDs (`wof.parent_id` , `wof.belongs_to` , and so on) as links but we aren't rendering `wof.id` as a link since there is no point in linking to the webpage we are already looking at.

Customizing things - Dictionaries

The second thing we needed to customize were the value of keys themselves. For example, we define concordances in Who's On First using short prefixes for other sources. A **Geonames** (<http://geonames.org>) ID becomes `gn:id`, a **Library of Congress** (<http://loc.gov/>) ID becomes `loc:id` and so on. That's useful and efficient for encoding data but not very satisfying to look at.

Just like text renderers, dictionary renderers are defined by invoking the `set_custom_renderers` method with "dict" as the first value and a custom function that returns a function (that returns a string) or `null`.

```
var dict_mappings = {
  'wof.concordances.gn:id': 'geonames',
  'wof.concordances.gp:id': 'geoplanet',
  'wof.concordances.loc:id': 'library of congress',
  // and so on...
};

var dict_renderers = function(d, ctx){

  if (dict_mappings[ctx]){
    return function(){
      return dict_mappings[ctx];
    };
  }

  return null;
};

mapzen.whosonfirst.yesnofix.set_custom_renderers('dict', dict_renderers);
```

In this example `loc:id` becomes "library of congress" and so on. You may noticed in the screenshots above that we haven't yet defined custom handlers for the `name:` properties so they all still get rendered with names like `eng_x_variant` or `chi_x_preferred`. We should fix that.

Customizing things - Exclusions (or locking things)

Finally some things just aren't up for debate. The reasons why an application may not want solicit feedback on certain bits are data are many and varied so we'll just leave it at at. The point is that you may *need* to prevent certain properties from being Yes No Fix -able, so you can.

You do this by invoking the `set_custom_exclusions` method padding “text” as the first argument and a custom function that returns a function (that return a boolean value). Like all the others, your custom function will be invoked with a `data` (the value) property and a `ctx` (the context or path) property. You might be starting to see a pattern by now.

```
var text_exclusions = function(d, ctx){

  return function(){

    if (ctx.match(/^geom/)){
      return true;
    }

    else if ((ctx.match(/^edtf/)) && (d == "uuuu")){
      return true;
    }

    // and so on...

    else {
      return false;
    }
  };
};

mapzen.whosonfirst.yesnofix.set_custom_exclusions('text', text_exclusions);
```

*In this example we are locking things prefixed with `geom` because their values are derived from **Sean Gilles** (<https://pypi.python.org/pypi/Shapely>)... I mean, math. We are also locking things prefixed `edtf` (for the Library of Congress’ **Extended Date/Time Format** (<https://loc.gov/standards/datetime/>)) whose value is already `uuuu` which is the shorthand for “unknown”.*

Customizing things – Report handlers

Yes No Fix doesn’t concern itself with what *happens* to a report, by design.

Its only concern is with the user controls for collecting assertions and finally rendering them in to a blob of CSV text. After that it is left up to individual applications using `yesnofix.js` to tell it what to do. An application might post the data to a remote server using an API, send the report somewhere via email, try to analyze the data locally and perform an action. Whatever.

Custom report handlers are just plain old Javascript functions that accept a string (the CSV text of the report) as their only argument, like this:

```
var submit_handler = function(report){  
    // do something with report here – the point is  
    // that yesnofix.js is agnostic about the meaning  
    // of "do something"  
};  
  
mapzen.whosonfirst.yesnofix.set_submit_handler(submit_handler);  
mapzen.whosonfirst.yesnofix.apply(data, target_el);
```

That's it. If your application has defined a custom submit handler it will be invoked when a user presses the `submit report` button. If you haven't defined a custom handler then the default behaviour is display the CSV text of the report in a new browser window.

Customizing things – enabling or disabling editing controls

For a little over a month now `yesnofix.js` has been running quietly on the Who's On First **Spelunker** (<https://whosonfirst.mapzen.com/spelunker/>), but disabled. What that means is that the properties displayed for a place are being rendered by `yesnofix.js` but the editorial controls have been turned off. We did that with a handy `enabled` method that accepts a boolean as its only argument.

```
mapzen.whosonfirst.yesnofix.enabled(false);
```

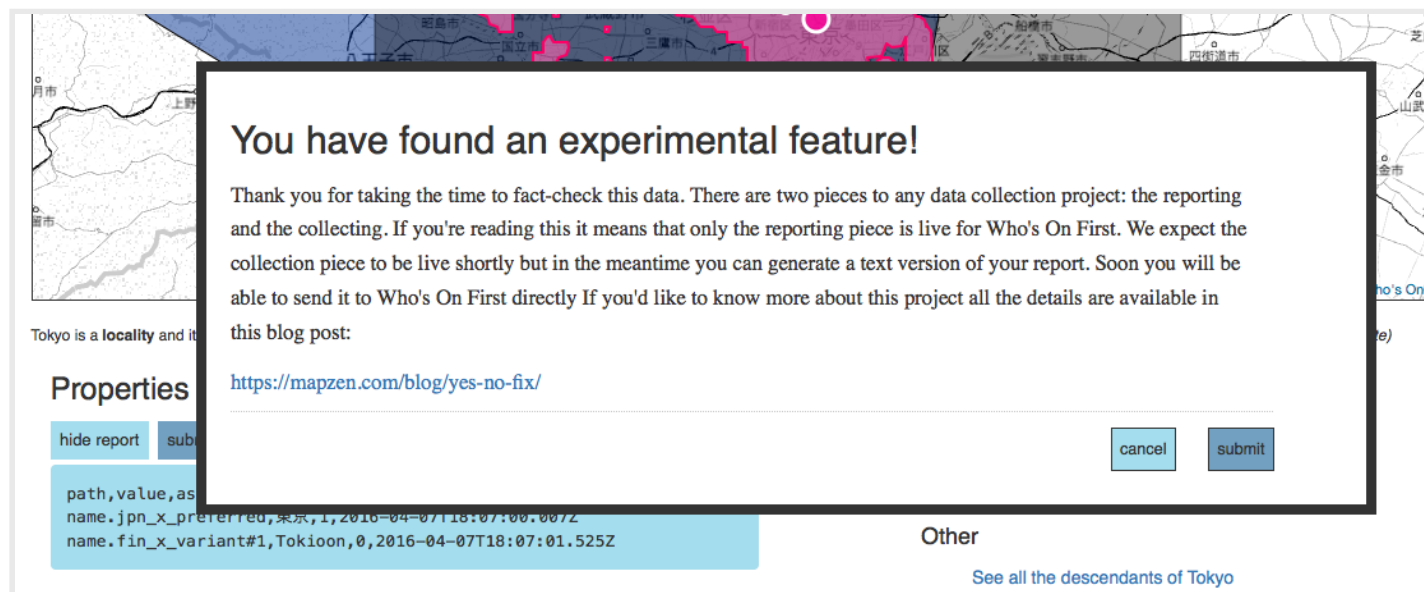
As we got ready for this blog post we simply removed that method call. For the technically minded, the Spelunker's **`mapzen.whosonfirst.properties.js`** (<https://github.com/whosonfirst/whosonfirst-www-spelunker/blob/master/www/static/javascript/mapzen.whosonfirst.properties.js>) file offers a concrete example of all the customizations described so far.

Yes No Fix and the Spelunker

I mentioned that Yes No Fix is now enabled on the Who's On First Spelunker, which is only half-true. Yes No Fix is absolutely enabled and we'd love for you to try and it out and **let us know** (<https://twitter.com/alloftheplaces>) what does and doesn't work. As of this writing though any

reports you generate *don't have a place (in Who's On First -land) to go yet.*

They will but, again by design, the data reporting is meant to be entirely separate from the data collection. We're finishing up the details on the data collection piece and so, in the meantime, we're going to light up the data reporting side of things and see how that works first.



Once both of these things are up and running the next task will be to use those reports to inform **the editorial tools and processes** (<https://github.com/whosonfirst/whosonfirst-www-boundaryissues>) we are starting to build for managing Who's On First. But that's another blog post for another day.

Yes No Fix is not just a Who's On First (or a Mapzen) thing



Yes No Fix is not designed to work with any one application but rather arbitrary blobs of data. For example, once we finish getting the data collection hooks working the obvious next step is to make the neighbourhoods (and other placetypes) in the Who's On First **I Am Here** (<https://mapzen.com/blog/iamhere/>) map Yes-No-Fix -able.

My current mornings-and-weekends project is building **a tool that will create a static HTML archive** (<https://github.com/thisisaaronland/go-cooperhewitt-shoebox>) of all the things I've collected at, or on the website of, the **Cooper Hewitt Smithsonian Design Museum** (<https://collection.cooperhewitt.org>). The tool uses **the museum's API** (<https://collection.cooperhewitt.org/api/>) to fetch data about **the objects** (<https://collection.cooperhewitt.org/api/methods/cooperhewitt.objects.getInfo>) I've collected as well as **the act of collecting** (<https://collection.cooperhewitt.org/api/methods/cooperhewitt.shoebox.items.getInfo>) them.

Like the properties in a Who's On First document the Cooper Hewitt data is encoded as **blobs of JSON** (<https://github.com/cooperhewitt/collection/blob/master/objects/184/915/01/18491501.js>

on) which is great for robots but not really great for humans to look at. I was able to drop the `yesnofix.js` libraries in to my code and **point the API response blobs at them** (<https://github.com/thisisaaronland/go-cooperhewitt-shoebox/blob/master/javascript/shoebox.item.js#L59-L81>) and *poof* there was something a little less terrible to look at. Just the way the Spelunker has been using `yesnofix.js` with the editorial controls disabled, for over a month now.



Lamp 1976-84-1

action	collect	
created	1460053888	
description		
id	97244129	
is_public	0	
lastmodified	1460053888	
refers_to	accession_number	1976-84-1
	creditline	
	date	
	decade	

This tool is still very much a “wet paint” project so it hasn’t been taught to use any of the fancy rendering tricks described above but that’s really just a question of time-and-typing. Likewise the actual Yes No Fix controls are disabled but maybe it’s worth enabling them and creating a custom submit handler directing people to send their reports to the museum via **their Zendesk account** (<https://cooperhewitt.zendesk.com/hc/en-us/requests/new>).

Cooper Hewitt, Smithsonian Design Museum > Submit a request

Submit a request

Your email address *

example@mapzen.com

Subject *


Object ID 18491501|

Description *

path,value,assertion,date
_global_dimensions_raw.depth#0,22.50,-1,2016-04-07T19:20:33.355Z

Please enter the details of your request. A member of our support staff will respond as soon as possible.

Attachments

 Add file or drop files here

This is what we mean when we say that Yes No Fix is designed to be agnostic about what happens with the data it collects. Also... **OMG, that lamp!!!!**
(<https://collection.cooperhewitt.org/objects/18491501/>)

Version Less-than-one

It is still early days for `yesnofix.js` and there are many things it doesn't do or doesn't do as well as it should yet. A short and not-comprehensive list includes:

- Keyboard shortcuts
- Working well (or at all) on mobile devices
- Working well with multi-level nested data structures, specifically where to *put* all those nested tables on a finite amount of screen space and then where squeeze in the editorial controls
- Support for any language besides English
- Proper testing for browser support
- Better documentation, particularly for all the customization handlers

But it's a start. It's also, we hope, a start at finding a middle ground between not accepting any feedback about the data in **Who's On First** (<https://whosonfirst.mapzen.com>) and throwing the doors wide-open and letting anyone edit whatever they want. The latter just isn't going to happen soon for a whole bunch of reasons but the former also feels kind of rude. Yes No Fix is meant to be a way for people to contribute to the data and, once the data collection piece is complete, for that work to have a safe place for that work to live on the internet and to start to be used to affect the final editorial work.

Yes No Fix is not a perfect solution to the problem, and there is plenty of work left to do, but our hope is that it will at least make things a little better than they were yesterday.

· 08 April 2016 ·



Aaron Straup Cope

Aaron is happiest in the presence of olive oil.

New Year's Resolutions – Money

targeted-editing (</tag/targeted-editing>) **osm** (</tag/osm>)

Maps are current as of Oct 2016.

A quarter of the way into 2016, have we reach the end of our little New Year's Resolutions series? I think so, but for the die hard resolutionists, I have one last post to inspire some edits.

It's time to count your pennies! Saving money is also a very popular New Year's resolution. If this was your top goal at the start of 2016, we hope you are on track. If you are just now remembering that you declared this resolution, you may need some inspiration. Start with adding the banks in your neighborhood and around your place of work. Banks are either loved or loathed, depending on fees and customer service. That leads to some very strong opinions which contain the requisite local knowledge for editing OpenStreetMap. Whether you love a bank or hate it, you definitely know where it is located!

Business listings and other points of interest are slowly gaining momentum in OpenStreetMap. Curious to know how prevalent banks are in OpenStreetmap? Check out this table which includes our New Year's Resolutions cities, summarizing banks with addresses, websites, and hours:

	Banks	with Addresses	with Website	with Hours
Auckland	214	5%	0%	0%
Dublin	204	22%	11%	6%
Hong Kong	536	4%	3%	3%
Melbourne	416	18%	56%	5%
Mexico City	323	6%	1%	5%
Mumbai	251	7%	1%	5%
San Francisco	436	39%	6%	4%

	Banks	with Addresses	with Website	with Hours
São Paulo	793	24%	1%	5%
Seoul	1,348	2%	1%	0%
Singapore	149	11%	1%	1%
Stockholm	200	13%	27%	19%
Vancouver	148	38%	5%	15%

***SQL queries (https://github.com/mapzen-data/targeted-editing/blob/gh-pages/queries/financial_sql.sql)** for those interested in generating stats for additional cities.

Looks like hours and websites aren't popular additions to OpenStreetMap banks in most cities, but perhaps that is OK. Generally speaking, banks typically have hours in between 9AM and 5PM or 6PM during the week in the United States, but there can be some variation in Saturday hours. All banks missing address tags would benefit from their addition. Addresses are well loved by geocoders and routing engines.

Helpful Tags

If you are behind in your savings goals, get inspired by adding or enhancing banks and ATMs in OpenStreetMap! Need some help choosing tags? Here are some suggestions which link to their corresponding OpenStreetMap wiki pages:

- **name=*** (<https://wiki.openstreetmap.org/wiki/Key:name>)
- **all appropriate address fields** (<http://wiki.openstreetmap.org/wiki/Addresses>)
- **website=*** (<https://wiki.openstreetmap.org/wiki/Key:website>)
- **operator=*** (<http://wiki.openstreetmap.org/wiki/Key:operator>) particularly helpful for ATMs
- **drive_through=*** (http://wiki.openstreetmap.org/wiki/Key:drive_through)
- **opening_hours=*** (https://wiki.openstreetmap.org/wiki/Key:opening_hours)
 - Tips (because the opening_hours tag can be confusing)
 - Use military time.
 - YES: 06:00-22:00
 - NO: 6am to 10pm
 - Use two letter day indicators: Su Mo Tu We Th Fr Sa
 - YES: Mo-Fr
 - NO: Mon thru Fri

- Do not add extra spaces between two letter day indicators or time ranges
 - YES: Mo-Fr 06:00-22:00
 - NO: Mo - Fr 06:00 - 22:00
- Separate more complex hours with a semicolon
 - YES: Mo-Fr 06:00-20:00; Sa-Su 08:00-16:00
 - No: another other separator besides a semicolon!
- Additional examples:
 - 24/7
 - Mo-Fr 06:00-22:00; Sa-Su 08:00-16:00
 - Mo-Su 06:00-22:00; Th off
- Looking for a helpful interface to help your format this tag properly? Check out **YoHours (<http://github.pavie.info/yohours/>)**
- Particularly helpful for ATMs
 - **amenity=atm (<http://wiki.openstreetmap.org/wiki/Tag:amenity%3Datm>)** includes a great list of tags, including ones that are critical for hearing and vision impaired patrons.

Where to Start

Banks are ubiquitous in many urban areas, and they may be central features in small towns. Spend some time noticing the banks you pass by on a regular basis, remembering that local knowledge is key when editing features in OpenStreetMap.

Interactive Map

Would it be helpful to see where the current banks and ATMs are in OpenStreetMap? There's a map for that! Hover over any of the bright blue highlighted features to bring up an info bubble with links to editing tools that can be used to add additional tags. (While you pan and zoom to your neighborhood, note that most banks show up at zoom 16, but you may have to zoom in more to see ATMs.)



Leaflet | Geocoding by Mapzen

This map is interactive! Open full screen ↗ (<https://mapzen-data.github.io/targeted-editing/te-banks/map/?bank>)

If you're not familiar with Seoul, open the **full screen map** (<https://mapzen-data.github.io/targeted-editing/te-banks/map/?bank>) and search for your city! Is a bank or ATM you are familiar with missing all together? Shift-click anywhere on the map to open iD or open your favorite OpenStreetMap editor to start mapping! For new features, start with the following tags:

- **amenity=bank** (<http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dbank>)
- **amenity=atm** (<http://wiki.openstreetmap.org/wiki/Tag:amenity%3Datm>)

If you are trying to decide between creating a point or a polygon to represent a feature, a point is great when the bank is in a building that contains other businesses and/or residences, and points are the preferred geometry type of ATMs. If the bank you would like to contribute occupies the entire building, digitize a building polygon if it doesn't already exist and add your tags to it.

Getting Started with OpenStreetMap

Need instructions on how to edit with iD? Here are some links to outstanding tutorials from LearnOSM, the OpenStreetMap wiki, and the United States Department of State's Humanitarian Information Unit:

- **LearnOSM** (<http://learnosm.org/en/>)
- **OSM Beginners' Guide** (http://wiki.openstreetmap.org/wiki/Beginners'_guide)
- **MapGive Learn to Map** (<http://mapgive.state.gov/learn-to-map/>)
- **Missing Maps Video Tutorials** (<http://www.missingmaps.org/contribute/#learn>)

Are you a mapping wiz and interested in a more advanced editor? Try out **JOSM** (<https://josm.openstreetmap.de>) with **excellent documentation** (<https://www.mapbox.com/blog/osm-mapping-guide/>) from Mapbox.

Editing with a Mobile Device

With points of interest, I bet you are interested in a mobile app to edit OpenStreetMap while you are on the go.

iOS users can check out **Go Map!!** (<https://appsto.re/us/dafwj.i>) by Bryce Cogswell. This free editor allows you to add features and edit existing features. Check out the **Go Map!!** (http://wiki.openstreetmap.org/wiki/Go_Map!!) wiki page for more details. **Pushpin** (<http://www.pushpinosm.org>) by Fulcrum is another excellent iOS editor for OpenStreetMap points of interest.

Looking for an Android equivalent? **Vespucci** (<https://play.google.com/store/apps/details?id=de.blau.android>) by Marcus Wolschon is a great option. Check out the **Vespucci** (<http://vespucci.io>) home page for documentation and tutorials.

Last but not least, for both iOS and Android, **MAPS.ME** (<http://maps.me/en/home>) now **supports editing** (<http://blog.maps.me/2016/04/lets-make-most-detailed-map-together.html>).

Get outside, increase your local knowledge, and share it with the world through your OpenStreetMap edits!

*Check out all the posts in the **Targeted Editing series** (<https://mapzen.com/tag/targeted-editing>):*

- **Airports** (<https://mapzen.com/blog/targeted-editing-airport-polygons>)
- **Banks** (<https://mapzen.com/blog/new-years-resolutions-money/>)
- **Building Names** (<https://mapzen.com/blog/targeted-editing-name-that-building>)

- ***Campus Mapping*** (<https://mapzen.com/blog/targeted-editing-campus-mapping/>)
- ***Cycleways*** (<https://mapzen.com/blog/targeted-editing-cycleways/>)
- ***Fitness*** (<https://mapzen.com/blog/new-years-resolutions-fitness>)
- ***Groceries*** (<https://mapzen.com/blog/new-years-resolutions-groceries>)
- ***Hospitals*** (<https://mapzen.com/blog/targeted-editing-hospital-polygons>)
- ***Schools*** (<https://mapzen.com/blog/targeted-editing-school-polygons>)
- ***Stadiums & Parking*** (<https://mapzen.com/blog/targeted-editing-tailgate-mania>)
- ***Street Names*** (<https://mapzen.com/blog/targeted-editing-no-name-roads>)
- ***Transit Colours*** (<https://mapzen.com/blog/targeted-editing-transit-colours>)
- ***Travel & Lodging*** (<https://mapzen.com/blog/new-years-resolutions-travel>)

· 12 April 2016 ·



Indy Hurt

Indy was our resident data scientist lending her geographic expertise to all things "open".

© 2017 Mapzen

State of the Map US Call for Proposals

osm (/tag/osm)

We have a lot of good things to say about **OpenStreetMap** (<https://mapzen.com/tag/osm/>) and the **State of the Map US conference** (<http://stateofthemap.us/>), and we think you do too! Why not **propose a talk** (<https://openstreetmap.us/2016/03/sotmus-cfp-2016/>) at SOTMUS in Seattle this July? The **deadline for proposals** (<https://openstreetmap.us/2016/03/sotmus-cfp-2016/>) is this **Sunday, April 17**.



Close Controls

[Leaflet](#)

(This map is interactive! Open full screen ↗ (<https://tangrams.github.io/bubble-wrap/#15/47.6089/-122.3212>))

See you in Seattle in July!

preview image via **SotMUS** (<http://stateofthemap.us/>)

· 13 April 2016 ·



Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.



John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

Missing the Point- GeoIP's, Points, Polygons, and a Precarious Farm in Kansas

whosonfirst (/tag/whosonfirst)



You Are Here ($\pm 10,000$ km)

There are few things more dangerous than an overconfident point when it's placed on a map. These sorts of points are intended to represent a place, like a house, a town, a city, or a country. But what happens when the thing not underneath that place is not that place (or is a fundamental misunderstanding of what that place is supposed to be)?

This past weekend, **Kashmir Hill** (<https://twitter.com/kashhill>) of **Fusion** (<https://fusion.net>) reported a **terrifying story** (<https://fusion.net/story/287592/internet-mapping-glitch-kansas-farm/>) that reflects the dark side of what happens when map data is misinterpreted and people are overconfident in what lies beneath that dot.

The story revolves around a small farm in Kansas that has been the victim of mistaken identity many times over. As Kashmir writes:

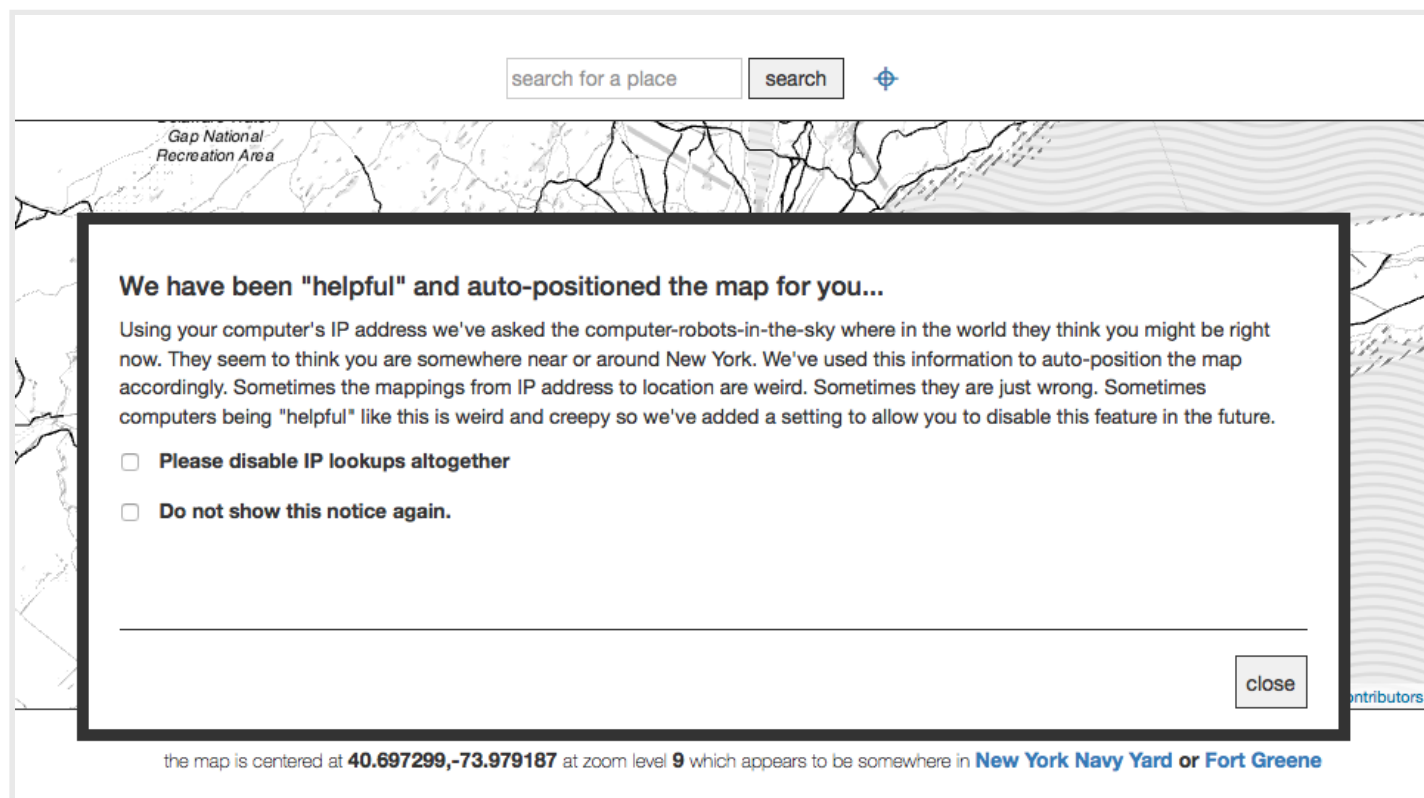
They've been accused of being identity thieves, spammers, scammers and fraudsters. They've gotten visited by FBI agents, federal marshals, IRS collectors, ambulances searching for suicidal veterans, and police officers searching for runaway children. They've found people scrounging around in their barn. The renters have been doxxed, their names and addresses posted on the internet by vigilantes. Once, someone left a broken toilet in the driveway as a strange, indefinite threat.

And to get to the underlying cause of this horrible case of mistaken identity is rather common technology called GeoIP. GeoIP providers attempt to connect clusters of IP addresses to a geographic region. Sometimes, that can be fairly specific, down to the city, other times it can only link to a state or country. If Facebook has asked you if you have tried to log in from another country you haven't visited, or a map that centers on your town by default, this is likely the technology that makes that possible. Sometimes, though, it centers on the wrong place, like when dial-up AOL had all their IP addresses coming from northern Virginia.

MaxMind, the most prominent GeoIP provider, only intended to give back the "general area" the IP address is in; not to indicate that the precise location lay beneath the pin. But that's exactly what many of the users of MaxMind's users have assumed that the data indicated. Which meant that any IP address that's known to be "Somewhere in America" (but can't be pinpointed to a specific city or state), MaxMind pointed right at this family farm. In some cases, these GeoIP leads can be useful, but when it all gets boiled down to *that point*, the nuance is often lost. And that can have drastic repercussions.

Part of the reason we're writing this is to point out that we have **our own project to augment the MaxMind GeoIP database** (<https://whosonfirst.mapzen.com/mmdb>) with data from **Who's On First** (<https://whosonfirst.mapzen.com/>) to interpret the results coming back from an IP lookup as a geographic area, and not a single point. Rather than sending back a point and some words saying where an IP address is location our modified version of the MaxMind database returns both a Who's on First ID and a bounding box (as well as its complete hierarchy)

for that location. It means the United States is the container for the United States and that small town in Kansas is just that small town in Kansas. This is still an experimental project and we are working through the mechanics of what we store in the databases (we could include complete polygons but that might makes things a bit... heavy) and how often things get updated. As of this writing we haven't yet updated our databases to **reflect the changes that MaxMind has made to their own data** (<http://fusion.net/story/290772/ip-mapping-maxmind-new-us-default-location/>) yet.



One of the other things this story has prompted us to do is finish up the work to enable IP lookups in **the I Am Here project** (<https://mapzen.com/blog/iamhere/>). When enabled the code on **the I Am Here website** (<https://whosonfirst.mapzen.com/iamhere>) will try to use your computer's IP address and its corresponding location (using a **Who's On First enabled MaxMind database** (<https://whosonfirst.mapzen.com/mmdb/>)) to automatically position the map. Previously the map would always load centered on San Francisco's majestic **Space Claw** (<https://www.flickr.com/people/spaceclaw/photosof/>)... I mean Sutro Tower, which is great but a bit tiresome if don't live in San Francisco and always need to start using the website by zooming out to a different location.

The reason that we haven't enabled the IP lookup functionality sooner is because we noticed that sometimes when you load the website from a computer in San Francisco the map would automatically position itself in... you got it, Kansas. Now when you load **I Am Here** (<https://whosonfirst.mapzen.com/iamhere/>) for the first time you will see a modal dialog

explaining that there are computers trying to be helpful, that sometimes their suggestions aren't very helpful and finally the option to tell the computers to stop helping you. Hopefully the computers will get it right more often than not and, because IP lookup is pretty cool when it works, there will be a way for people to use I Am Here without having to always start in San Francisco.

But this story, and **others** (<http://fusion.net/story/214995/find-my-phone-apps-lead-to-wrong-home/>) like it (<https://gimletmedia.com/episode/53-in-the-desert/>) should be a wake-up call to folks who design geospatial systems. When conveying ambiguity, it's hard to think how your users' users will necessarily interpret the thing. And while longitude and latitude are ubiquitous, they're not always right to send along the context involved.

MaxMind has **already taken some corrective steps** (<http://fusion.net/story/290772/ip-mapping-maxmind-new-us-default-location/>) by changing the point that represents America from the farm at [38,-97] to the center of a nearby lake, but it still doesn't address the necessary issue of conveying how big a place match may be.

But to them, and to you, please: Consider the polygon.

· 14 April 2016 ·



David Riordan

Former product manager for Mapzen Search. Historical mapping, open tools, open access, and open data.



Aaron Straup Cope

Aaron is happiest in the presence of olive oil.

Who's on First is now powering Mapzen Search

[search](#) (/tag/search) [whosonfirst](#) (/tag/whosonfirst)

We made a big change to **Mapzen Search** (<https://mapzen.com/search>) last week. Didn't notice? Exactly!

Since last Friday, Mapzen Search is being powered by **Who's on First** (<https://whosonfirst.mapzen.com>), Mapzen's open data gazetteer. Now, we can make changes to cities, countries, neighbourhoods, and borders in days rather than months, and take advantage of new data types as the *Who's on First* team adds it (not to tease postal codes **but...** (<https://github.com/whosonfirst-data/whosonfirst-data-postalcode>)).

There have been a myriad of changes under the hood, with subtle improvements in Search and Autocomplete, and many more to come in the next few weeks.

There have been a few additions and clarifications to **our documentation** (<https://mapzen.com/documentation/search>), so be sure to check out the latest there.

What's on next?

· 14 April 2016 ·



David Riordan

Former product manager for Mapzen Search. Historical mapping, open tools, open access, and open data.

Search Endpoint

search (/tag/search)

For a little under a year, I've had the pleasure of serving as product manager for **Mapzen Search** (<https://mapzen.com/projects/search>), working with some of the most brilliant folks working in mapping today as we build a world-class **open source geocoder** (<https://github.com/pelias/pelias>)¹ and a remarkable service open to all on top of it. Not to mention playing a small part in many of the other incredible things we do at Mapzen. So it's incredibly bittersweet for me to share that I'm stepping way from the Github issues and leaving Mapzen for the **Brown Institute for Media Innovation** (<http://brown.columbia.edu/>), a joint research center between Columbia School of Journalism (where I'll be based) and Stanford's School of Engineering, as their Chief Technology Innovation Officer.



When I joined Mapzen, it was because I was getting the chance to join a remarkable team, working to create a thing I thought needed to exist. I wanted a better open source geocoder because I wanted to use it on my own data, well really the data of **many New York Cities past** (<http://spacetime.nypl.org>). My former team at The New York Public Library needed a geocoder for historical New York City, and **Pelias** (<https://github.com/pelias/pelias>) seemed like the right tool to make that happen. We haven't built an open source historical geocoder out of Pelias², but we've been working on a whole lot more that makes world class geodata accessible to everyone.

We created **Who's on First** (<https://whosonfirst.mapzen.com>), an open data gazetteer that's already the single best place to list and connect **all of the places** (<https://mobile.twitter.com/alloftheplaces>).

We sponsored **Libpostal** (<https://github.com/openvenues/libpostal>), the first open source worldwide address parser based on statistical learning that can make sense of nearly any address from anywhere.

We built **Eraser Map** (<https://mapzen.com/blog/erasermap-beta/>), a mapping app that *doesn't track you*, a remarkable achievement.

But Mapzen's always been about more than just writing open source mapping software. There had been plenty of good open source mapping software before OpenStreetMap just miracled into existence. But what OpenStreetMap catalyzed was the synthesis of open source and open data created by open communities. Now, for the first time, there were opportunities for people to map the world, to create the tools to create the data that those communities could collect. And for the first time, companies and regular people could collaborate across open data projects to do what had before been the sole jurisdiction of gods, kings, and generals: to try and see the whole world. And for the first time there was the ability for people to use these tools for self-representation.

What brought me to Mapzen wasn't just the prospect of building these tools, but being a part of a place that wanted to make geo more open forever. Not just so that it's free in cost, but free for all to use to advance research, to tell stories, and to be preserved so we can remember our present in the future. It's all set forth in our **Magna Carto** (<https://mapzen.com/blog/our-magna-carto>).

We've come a long way over this past year. And I can't wait to use what we've made. I suspect a lot of others will as well. But the truth is, this doesn't happen alone. Mapzen's all about building communities to build open data and open tools. They want to do it with you. And I'm excited to help from the outside. Hopefully you are too. The work we do here means the world to me. Literally.

Onward, together. This is going to be fun!

p.s.

You know where to find me:

- **@riordan** (<https://twitter.com/riordan>)
- **email** (<mailto:dr@daveriordan.com>)

Image: Manuscripts and Archives Division, The New York Public Library. (1959 - 1971). *Aerial View of Perisphere* Retrieved from <http://digitalcollections.nypl.org/items/917c3c71-b8ff-5589-e040-e00a18067ba2> (<http://digitalcollections.nypl.org/items/917c3c71-b8ff-5589-e040-e00a18067ba2>)

1. Yes, *build*; current tense. It's still a work in progress. Building a machine that can find every place in the world is objectively a hard problem, so no, we're not done yet. ↩
2. ...yet. ↩

· 15 April 2016 ·



David Riordan

Former product manager for Mapzen Search. Historical mapping, open tools, open access, and open data.

© 2017 Mapzen

Introducing the RouteStopPattern

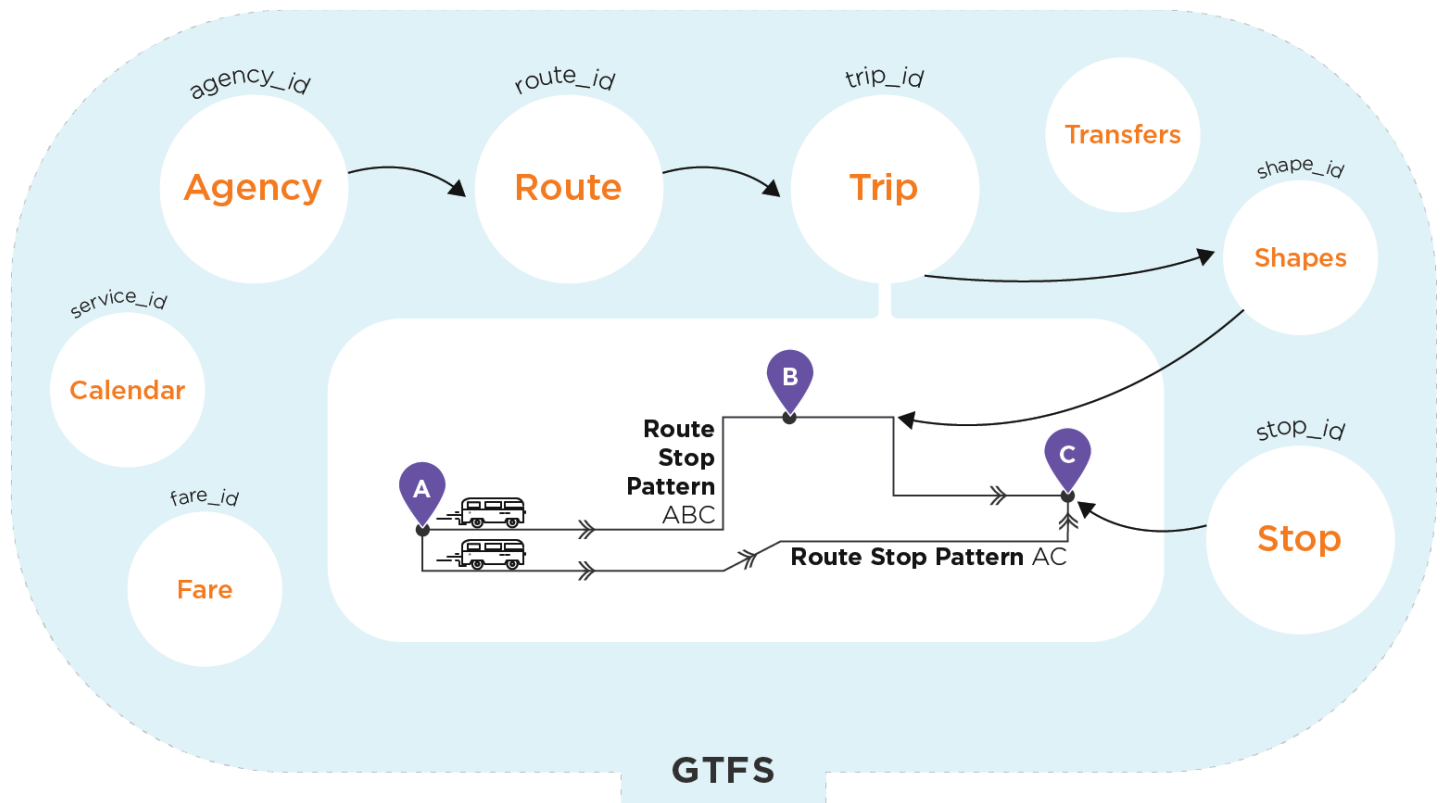
transitland (/tag/transitland)

Think of the last time you hopped on a bus, train, or ferry in a new place. Chances are you came across a map that displayed the right path, using the right route, and maybe just the right time to get from your departure to your destination. But where does that path come from? The need to ask this question may not seem obvious if all transit routes looked something like **this** (http://www.actransit.org/pdf/maps/version_31/51A.pdf). But what if the route you needed to take had more than two directions? Or had stops that were not always served after their previous stops? What if the whole route looked something like **this** ([http://www.cttransit.com/Uploads_RTMaps/nh_Bcongress_map\(11\).pdf](http://www.cttransit.com/Uploads_RTMaps/nh_Bcongress_map(11).pdf))?

So, what exactly is a route?

Within transit data, the route is front and center. Highly recognizable, **often colorful** (</blog/targeted-editing-transit-colours/>), and physically tangible, routes are used by transit operators to organize service, and by passengers to compartmentalize journeys. Behind the scenes, a transit operator publishing its data using the **General Transit Feed Specification** (<https://transit.land/documentation/glossary/#gtfs>) understands routes in terms of individual components called trips.

Each trip is a sequence of stops with associated arrival and departure times. A trip has an associated geographical line, called a shape, that describes a path connecting one stop with another stop, and wherever else a vehicle might visit (such as a bus yard). A trip may serve any subset of stops within a route, so a stop in a route may be served by one trip, and not another. The true definition of a transit route, then, is a collection of lines within a particular geographical region consistently and cohesively served by trips.



The RouteStopPattern: Transitland Datastore's missing link




Transitland takes GTFS trips and shapes and transforms them into something a little different: the **RouteStopPattern** (<https://transit.land/documentation/datastore/routes-and-route-stop-patterns.html>). At its core, the RouteStopPattern is the geometric representation of each of the unique combinations, within a single route, of trip stop sequences together with shape lines. Very commonly, multiple trips running at different times of the day will share the exact same stop sequence and single shape, and will therefore be represented by a single RouteStopPattern. Transitland has also made RouteStopPatterns available from its Datastore API. As you can see from **this query** (https://transit.land/api/v1/route_stop_patterns.json?onestop_id=r-dr5r7-statenislandferry-b860bb-38447b), RouteStopPatterns contain the `stop_pattern` array, corresponding to the stop sequences present in the original trips, a `geometry` corresponding to the shape line, the `route` which the RouteStopPattern belongs to, and the `identifiers` and `trips` arrays which point back to the GTFS shape and trips.

Within the Transitland data model, RouteStopPatterns are accessible from both **Routes** (https://transit.land/api/v1/routes.json?onestop_id=r-dr5r7-statenislandferry) and **ScheduleStopPairs** (https://transit.land/api/v1/schedule_stop_pairs.json?route_onestop_id=r-dr5r7-statenislandferry). Since ScheduleStopPairs are derived from a

single trip, they will reference only one RouteStopPattern. With these two links, RouteStopPatterns can now be combined to form a whole route, or used individually to show the true path a vehicle might take between stops at certain times.

For an interactive visualization and mapping that helps understand the Transitland data hierarchy from Operators to Routes to RouteStopPatterns to Stops, head over to: <http://transitland.github.io/route-spotter/> (<http://transitland.github.io/route-spotter/>)

transitland



Route Spotter

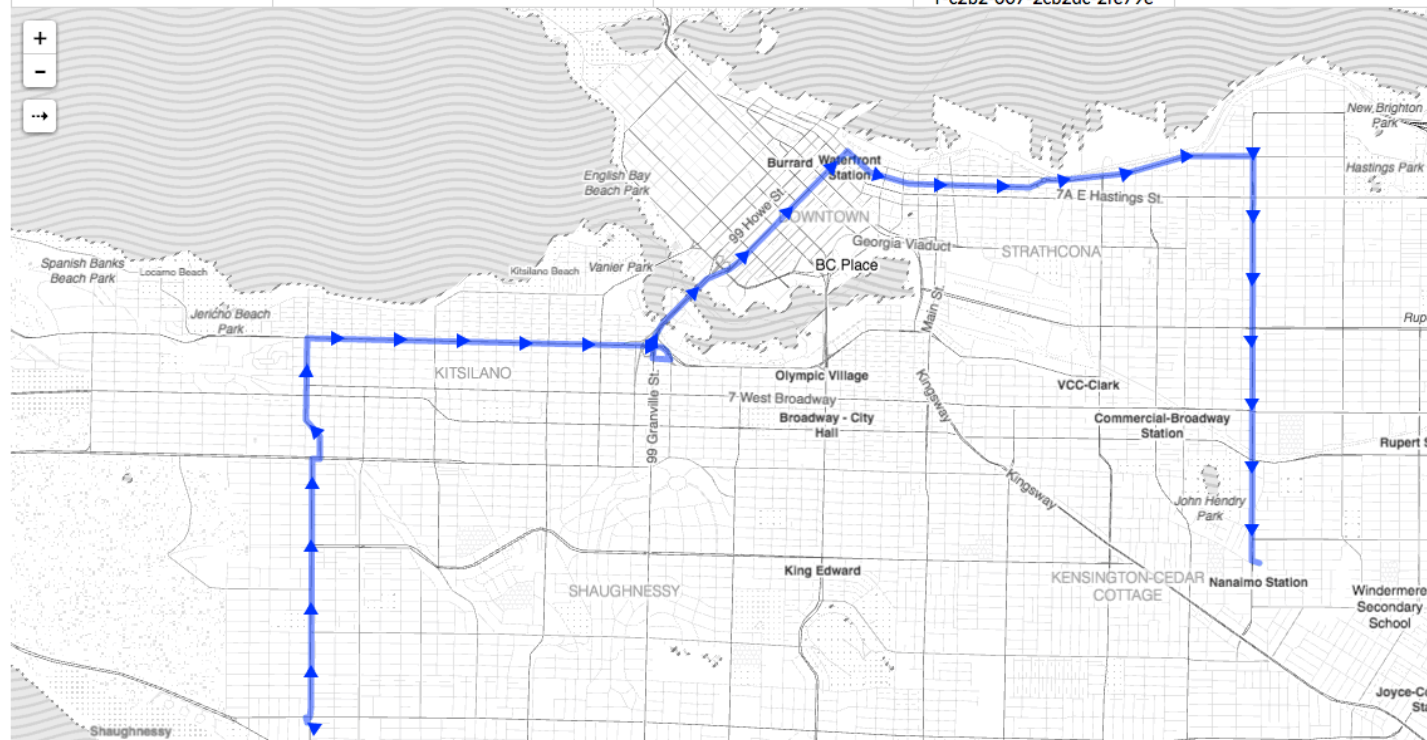
Click on a region below to start drilling down through the the contents of the Transitland Datastore. Each **Route** can have multiple **RouteStopPatterns** to represent the unique orders in which stops are visited and the roads or tracks taken between those stops. At each stop, we've estimated the distance already traveled along the route path. To learn more, see the [Transitland Datastore documentation](#).

Region	Operator	Route	RouteStopPattern	Stop
Irento	TransLink (o-c28-translink) West Coast Express (o-c2b-westcoastexpr... 	003 (r-c2b2n-003)		s-c2b2hffu7d-dunbai
Trinity County		004 (r-c2b2-004)	r-c2b2-007-427868-a6f3db	s-c2b2hfr2j-nbdunbarst
Vancouver		005 (r-c2b2q-005)	r-c2b2-007-d207d5-640ad7	s-c2b2hg72b6-nbdunbars
Victoria		006 (r-c2b2q-006)	r-c2b2-007-15e412-00ee41	s-c2b2hge2c1-nbdunbarst
Wallowa County		007 (r-c2b2-007)	r-c2b2-007-ec2873-63c0b2	s-c2b2hgg6cv-nbdunbarst
Washington		008 (r-c2b2n-008)	r-c2b2-007-4a531d-00adfc	s-c2b2hu5m9q-nbdunbars
			r-c2b2-007-2cb2dc-2fe79e	

+

-

→



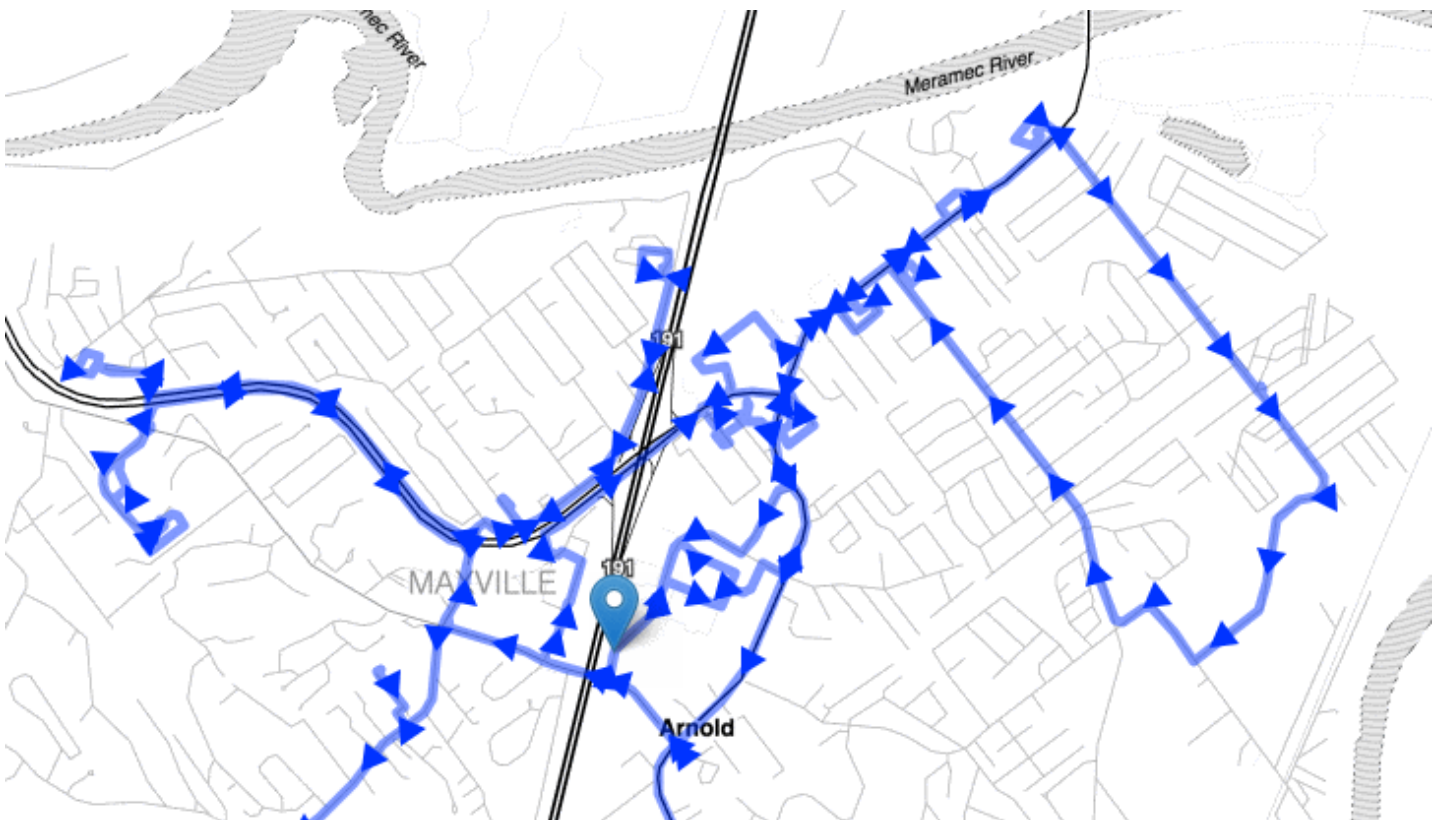
Leaflet | Tangram | © OSM cc

Going the Distance

But there's more! GTFS also allows a field called `shape_distance_traveled` to be populated by the distance of stops along the associated trip shape line. In general, this value is important in relating a stop to its physical location along the trip, as stops are not always found on the shape line. It is especially useful, however, for routing applications, since the distances between stops can become travel distances or even be used to estimate travel time.

Unfortunately, we have found that the vast majority of GTFS feeds do not have the `shape_dist_traveled` field populated. Therefore Transitland has developed **an algorithm** (<https://transit.land/documentation/datastore/routes-and-route-stop-patterns.html>) to infer these distance values from the shapes and stops given. It is still a work in progress, especially given the complex nature of transit routes (Remember those loops? Turns out they make life complicated for our digital yardstick). In any case you can find the distances in an array named `stop_distances` in the `RouteStopPattern`, or in `ScheduleStopPair` as `origin_dist_traveled` and `destination_dist_traveled` for the origin and destination stops.

Can't Stop Won't Stop



The data in GTFS is complex, and as we refine these tools we are working on more visualization techniques, including animations – these will help clarify complex and overlapping route topologies. Stay tuned!

· 15 April 2016 ·



Alex Smith

© 2017 Mapzen

Mapping with Perspective – April 26 at Mapzen's San Francisco office

Mapzen will be hosting an evening of twisted talks on mapping perspective at our San Francisco office on Tuesday, April 26—otherwise known as **National Pretzel Day**

(<http://www.holidayinsights.com/moreholidays/April/nationalpretzelday.htm>). We all know maps aren't real life but a salty interpretation of space. We'll be celebrating that power of interpretation with a stellar lineup of speakers. From Charlie Loyd's view from above (Mapbox Satellite) to Georgina Voss' Situated Systems of below, along with the surface view of Danny Whalen's buses (Remix) to the boundary tension of Aaron Cope's Who's On First gazetteer, we'll braid good talk and geo for a night of many perspectives. **Come join us** (<https://mapzen.splashthat.com>)!

Refreshments, snacks, and beautiful views of the Bay and Embarcadero will be served. **RSVP** (<https://mapzen.splashthat.com>) and learn more at <https://mapzen.splashthat.com> (<https://mapzen.splashthat.com>).



Actual view!

Speakers

Aaron Straup Cope

Aaron is currently Editor at Large and the creator of the Who's On First project at Mapzen. Between 2012 and 2015 he was Head of Engineering at the Cooper Hewitt Smithsonian Design Museum, responsible for the architecture, implementation and general table-pounding of the museum's digital infrastructure and The Pen. Before all of that, Aaron was Senior Engineer at Flickr focusing on all things geo, machinetag and galleries related between 2004 and 2009. From 2009 to 2011 he was Design Technologist and Director of Inappropriate Project Names at Stamen Design. Aaron does not normally speak in the third person and is happiest in the presence of olive oil.

Charlie Loyd

Charlie is a programmer and writer from the Pacific Northwest. He works with satellite imagery at Mapbox, on projects like cloud removal and artifact reduction. Beyond image processing, his curiosities include psychogeography, ecology, spaceflight, the Pacific Ocean, typography, the historiography of conflict, dataviz, and geoengineering. He is not interested in architecture, cities, or the intersection of technology, art, and culture.

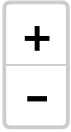
Georgina Voss

Georgina is a co-founder of research consultancy Strange Telemetry, a member of the Experimental Research Lab at Autodesk's Pier 9 facilities, and teaches at Goldsmiths, University of London. She leads complex research and design projects, writes, lectures, and makes artworks about the intersection of technology and politics.

Danny Whalen

Danny is an engineer, data geek, and friendly human. He co-founded Remix, a company helping cities around the world plan better public transit. Danny was previously a fellow at Code for America working with the GIS group in the city of Charlotte, NC. Don't hesitate to get in touch with him if you want to chat about bus systems, software development, or data visualization.

Join us

[Leaflet](#)

(*This map is interactive! Open full screen ↗* (<https://tangrams.github.io/bubble-wrap/#17/37.79117/-122.39187>))

- When: Tuesday, April 26, 6:30 - 8:30 pm
- Where: 201 Spear Street, 16th Floor, San Francisco, CA 94105 (closest transit station: Embarcadero)
- Why: Because maps always tell a story

Let us know you're coming! RSVP and more information at <https://mapzen.splashthat.com> (<https://mapzen.splashthat.com>).



MAPZEN

TUESDAY APRIL 26 AT 6:30PM

MAPPING WITH PERSPECTIVE

RSVP



· 18 April 2016 ·



Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.

© 2017 Mapzen

CartoDB & Mapzen are in an open relationship

vector-tiles (/tag/vector-tiles) **search** (/tag/search)

Mapzen is excited to be partnering with CartoDB to bring our world-class vector tiles, navigation, and geocoding to all of their customers.

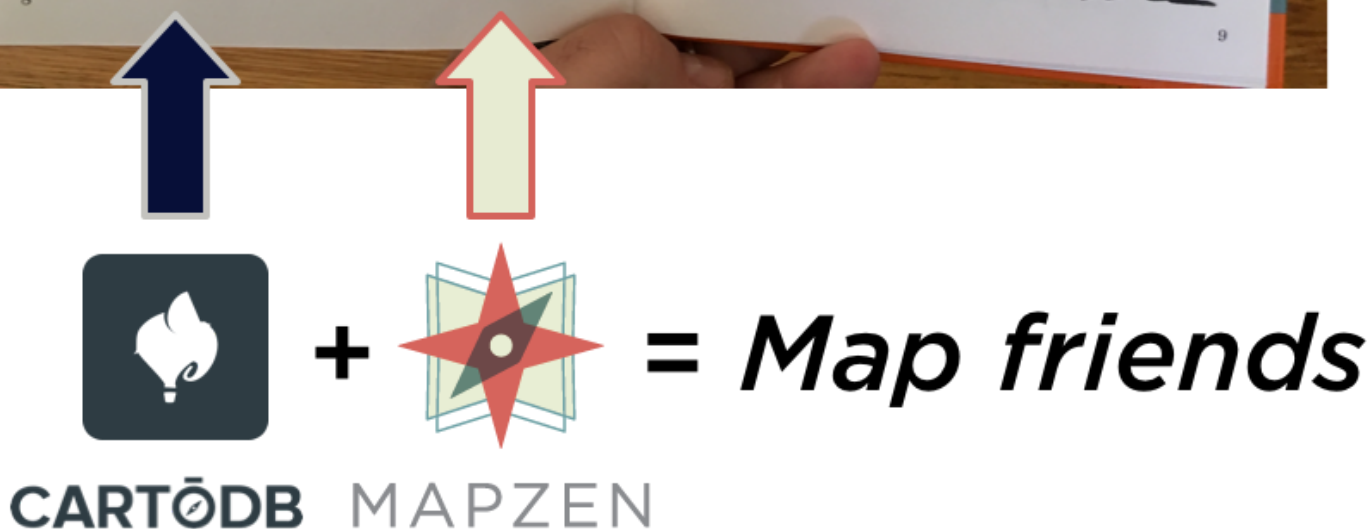
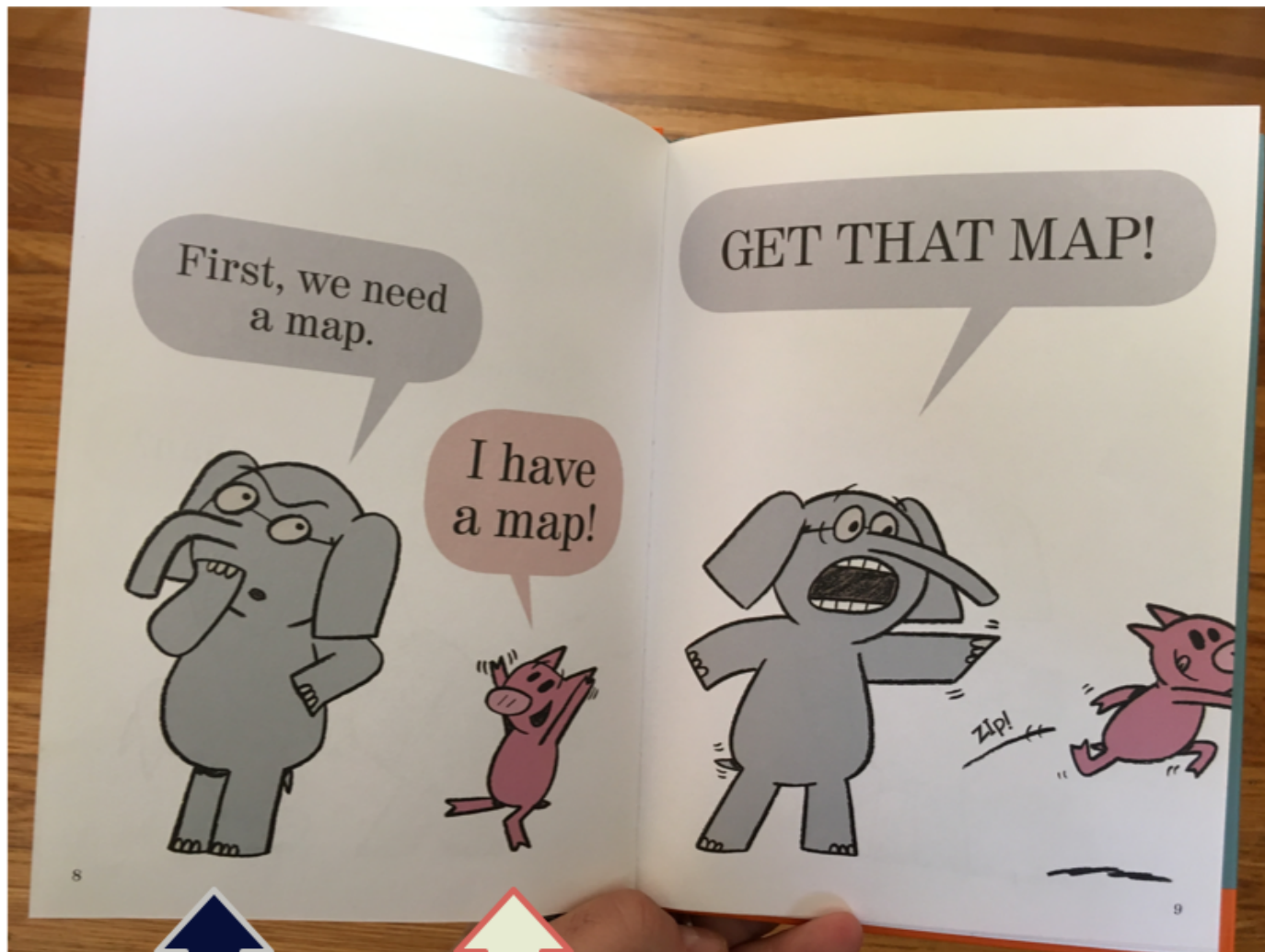


We were founded on the premise that open data would **irrevocably change the business of geospatial software** (<https://mapzen.com/blog/our-magna-carto/>). Over the past two and a half years we've built a complete set of open source tools that we've always known could go head-to-head with commercial mapping platforms. It's no surprise that our friends at CartoDB, **fellow proponents of open source** (<https://github.com/CartoDB/cartodb>), recognize this and are the first to integrate Mapzen technology into their **Location Data Services** (<https://cartodb.com/location-data-services>) product.

We're excited that *every* CartoDB user will gain access to these valuable tools at a scale previously unavailable: customers on their **Enterprise plan** (<https://cartodb.com/enterprise/>) will get custom limits and quotas for geocoding and routing, along with limitless views of maps.

Mapzen will continue to deliver a generous free tier for individual developers and small businesses. For customers requiring high quotas, service behind firewalls, SLAs, enterprise support, or professional services, CartoDB is ready to help with your business plan – **drop them a line** (<https://cartodb.com/enterprise/>)!

Our suite of geo and map tools is great, and is only getting better. We are very excited that CartoDB and their friends **get to share the smooth ride along the open road, map in hand** (<https://www.kirkusreviews.com/book-reviews/mo-willems/lets-go-for-drive/>).



Here we see a dramatic recreation of the negotiation process.

*Map! Map!
Mappy-map-
map!*



Images from “Let’s Go For A Drive” by Mo Willems (<https://www.kirkusreviews.com/book-reviews/mo-willems/lets-go-for-drive/>)

· 21 April 2016 ·



Randy Meech

Billerica Memorial High School graduate. BA, MTS. Mapzen CEO 2013-present.



Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.

© 2017 Mapzen

A high note for release notes

search (/tag/search)



photo via **draket** (<https://www.flickr.com/photos/draket/10055170993>), (CC BY-ND 2.0)

The search team wants to sing forth a confession:

♪♪ We're not awesome at everything, like everyone assumes. ♪♪

There, we said it! WOW, that's a relief! There is one thing we've been particularly *not* awesome about in the past: clearly articulating to our users the changes that have been regularly deployed to our hosted Mapzen Search service. Our new **Release Notes**

(<https://mapzen.com/documentation/search/release-notes/>) page shows we're much better at that now, but as with any problem, the first step was admitting we had one.

To give a bit of background, let's quickly examine the current underlying framework at the core of Mapzen Search. We are continuously working on an open-source geocoding engine we lovingly call **Pelias** (<https://github.com/pelias/pelias>). Pelias is the collection of software modules that work together to turn text into map coordinates, and vice versa. Code is wonderful, and *open-source* code means everyone can see how the geocoding sausage is made. However, code alone is not enough to make the solution accessible, because not everyone can read code, or understand how to deploy that code on a machine. So to allow everyone equally easy access to what we're building, we're also hosting an instance of Pelias. That hosted instance is known as **Mapzen Search**, and all you need to leverage geocoding in your work is a free **Mapzen Developer Account** and a search api key, both of which can be obtained **here** (<https://mapzen.com/developers>).

Now back to our admitted problem of not effectively communicating every release. From this revelation came the creation of a new heading in our documentation, appropriately named, you guessed it, **Release Notes** (<https://mapzen.com/documentation/search/release-notes/>). It's a place for documenting changes in Pelias that roll out to Mapzen Search, any and every time changes are made. The changes called out in this high-level document are intended to inform users of our Mapzen Search service about changes they can expect to see. We promise to write them in plain English, and explain how the changes in our code will translate to an improved experience for our users, and our users' users. Self-improvement efforts on the technical release notes for the underlying Pelias geocoding engine are still underway and we will report back soon.

We hope our new efforts in communicating more effectively will help our users have more insight into their favorite open-source geocoding service. We also hope it spurs conversation around functionality, as well as letting users discover new features as soon as they are rolled out, instead of finding them by chance months later in the documentation.

Please bookmark the **Release Notes** (<https://mapzen.com/documentation/search/release-notes/>) page and check in every week or so for exciting installments in the life of Mapzen Search.

· 22 April 2016 ·

Diana Shkolnikov

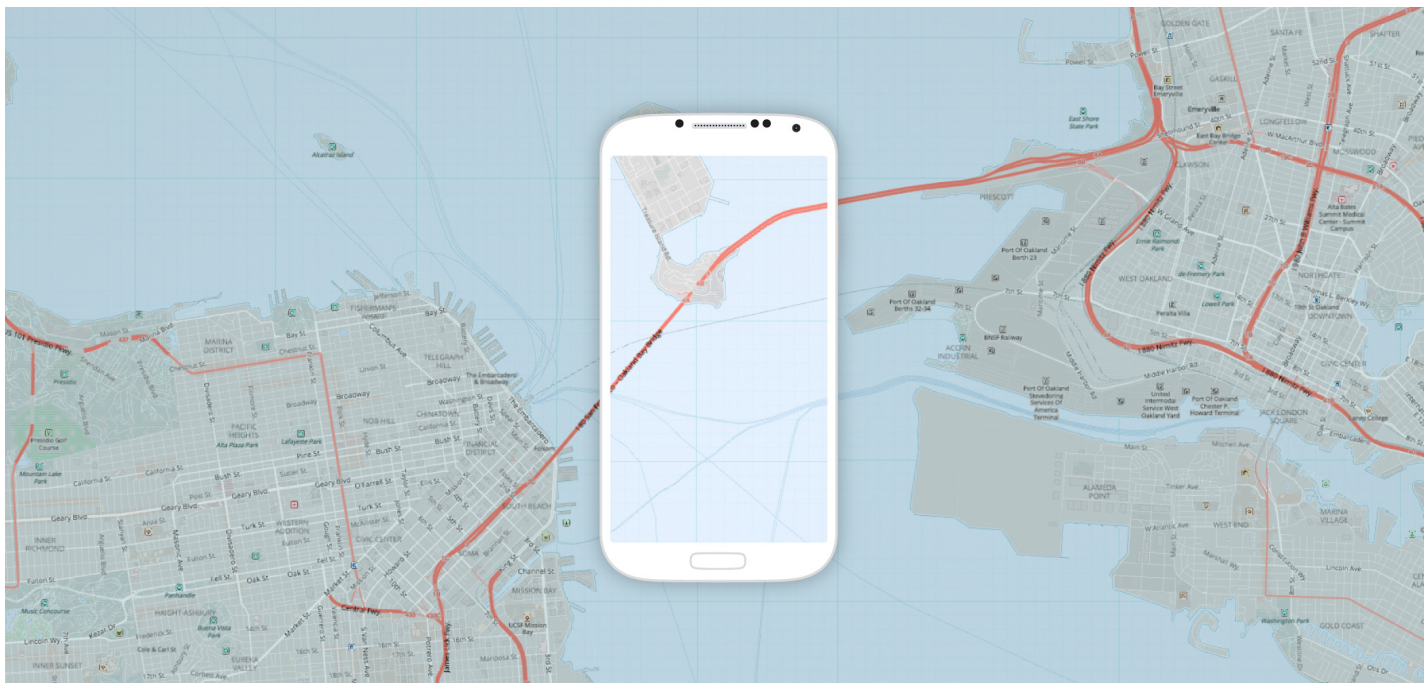


Diana is the engineering director of Search@Mapzen and a proponent of mandatory fun, testing, education, and paleo, not necessarily in that order.

© 2017 Mapzen

We launched an Android SDK

mobile (/tag/mobile) android (/tag/android)



If you were lucky enough to have been at the Samsung Developers conference you may have heard that we launched our Android SDK, but if you weren't there...

We launched our Android SDK!!!

Our Android SDK is meant to be a thin wrapper that packages up the Mapzen services and makes them easy to incorporate into your app. Currently the Mapzen SDK features map rendering and location tracking. You can easily **draw lines** (<https://mapzen.com/documentation/android/polyline/>), **drop pins** (<https://mapzen.com/documentation/android/markers/>), **change map styles** (<https://mapzen.com/documentation/android/styles/>), and **move the camera around** (<https://mapzen.com/documentation/android/basic-functions/>) though the Mapzen SDK or dig directly into **Tangram** (<https://mapzen.com/documentation/tangram/>) for more advanced options. Also, because we've incorporated our **L.O.S.T** (<https://github.com/mapzen/lost>) project there are no other mapping or location services dependencies required (seriously).

Over the next few months we will be incorporating navigation and search into the SDK through the following Mapzen projects:

- **On the road** (<https://github.com/mapzen/on-the-road>) - The Mapzen Turn-by-Turn wrapper and other routing utilities
- **Pelias** (<https://github.com/pelias/pelias-android-sdk>) - The Mapzen Search wrapper

Try out the **sample app** (<https://github.com/mapzen/android/tree/master/sample>) or **visit Github** (<https://github.com/mapzen/android>) to learn more about our Android SDK and follow along with our development.

· 27 April 2016 ·



Mike Cunningham

I'm Street View famous in two cities and I do product @mapzen.



Chuck Greb

Chuck is an open source enthusiast, test-driven evangelist, and clean code connoisseur of all things mobile.



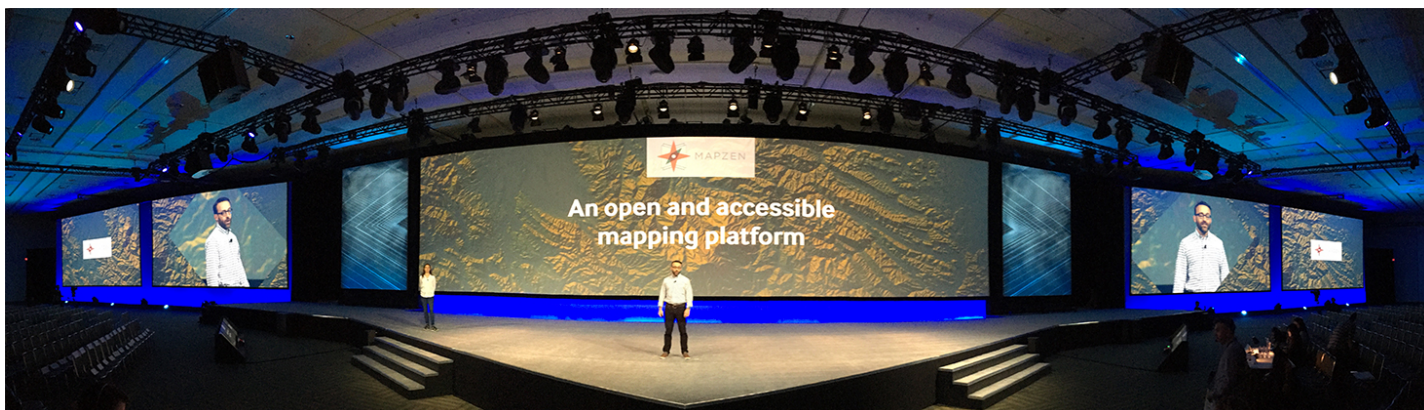
Sarah Lensing

mobile developer. startup junkie. athlete. older sister. nyu alum. vermonter. maker of beef jerky and other random food items.

© 2017 Mapzen

Samsung Developer Conference

Randy's talking about Mapzen during the keynote at the Samsung Developer Conference. (Here he is on the big screen(s), practicing.)



If you're at the show, stop by our booth and say hello!

If you missed it, you can take a look at his slides, **with** (<https://dl.dropboxusercontent.com/u/57592475/mapzen-sdc-audio.m4v>) or **without** (<https://dl.dropboxusercontent.com/u/57592475/mapzen-sdc-silent.m4v>) sound.

Also, we announced our Android SDK! **Learn more** (<https://mapzen.com/blog/android-sdk>), and let us know how you use it in your apps!

· 27 April 2016 ·



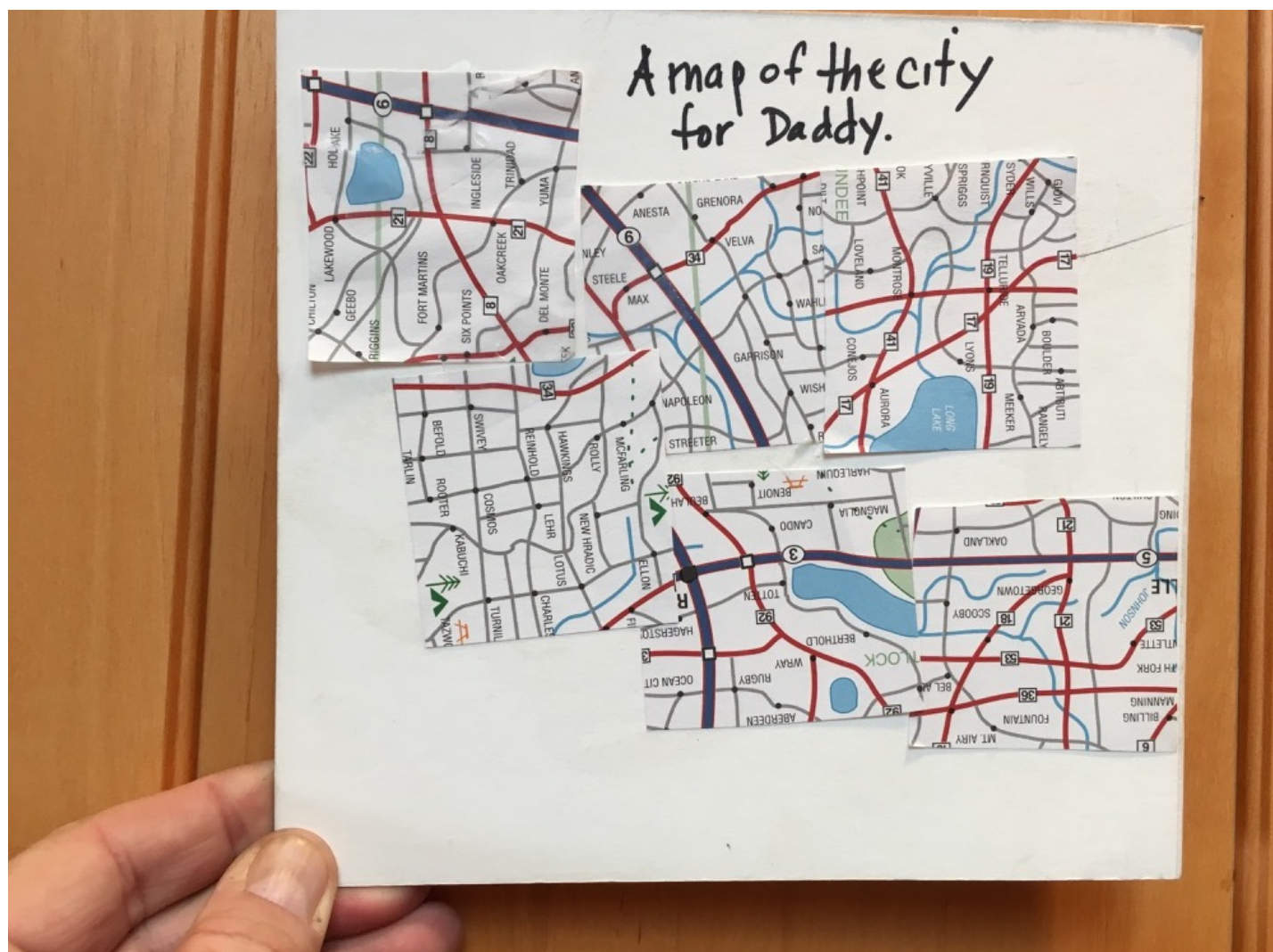
Randy Meech

Billerica Memorial High School graduate. BA, MTS. Mapzen CEO 2013-present.

The Kids at Mapzen

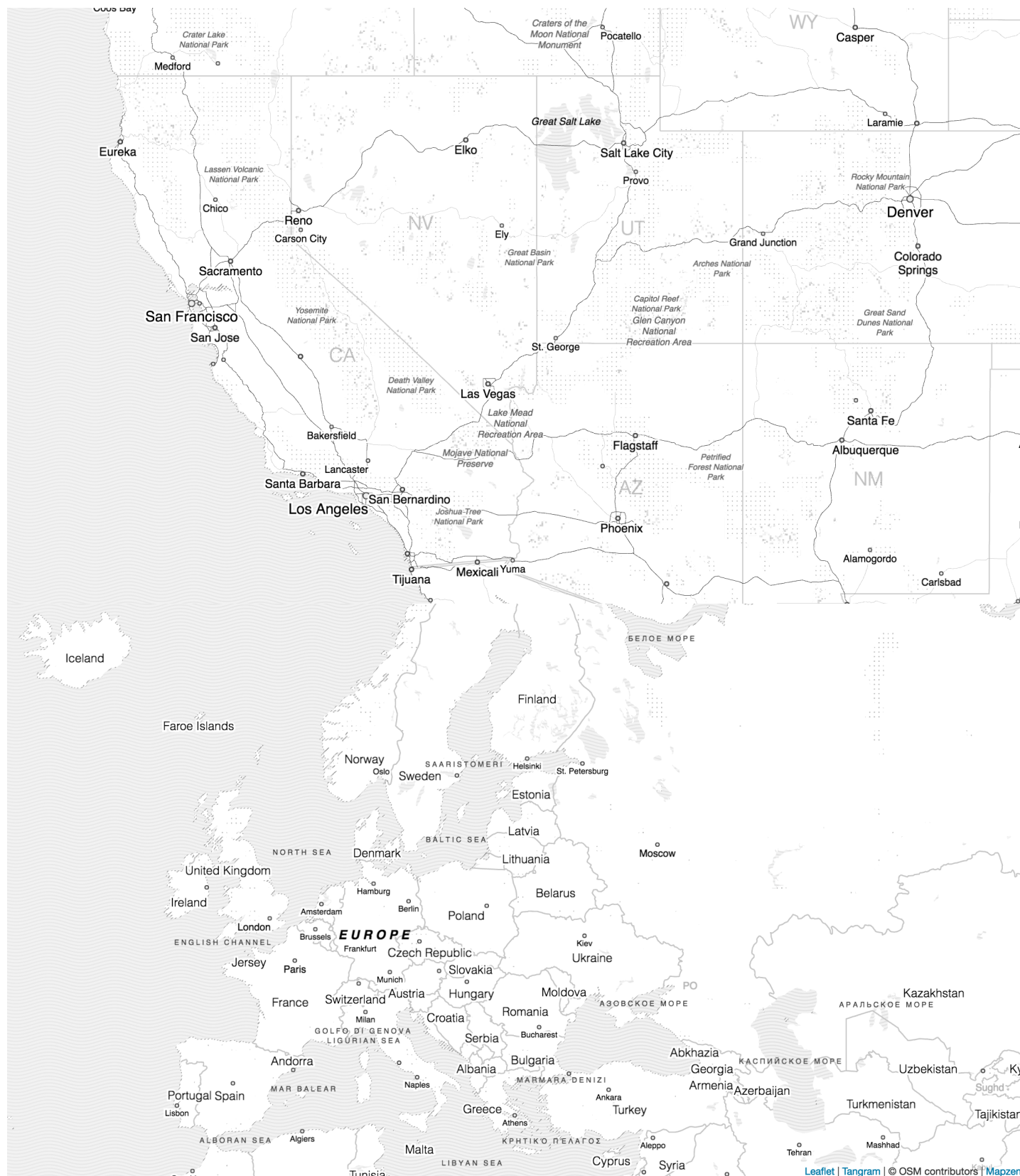
Mapzen is not your typical startup. This is reflected **in our philosophy** (<https://mapzen.com/blog/our-magna-carto/>), but more importantly by the people who work here.

But sometimes we even surprise ourselves – as we got closer to *Take Your Kids to Work Day*, we started counting how many kids the 46 of us at Mapzen have: we were kind of amazed to learn there are 30! And 4 grandkids! This is pretty high for a startup. For TYKTWD, many of us brought our kids to our offices and made some maps.



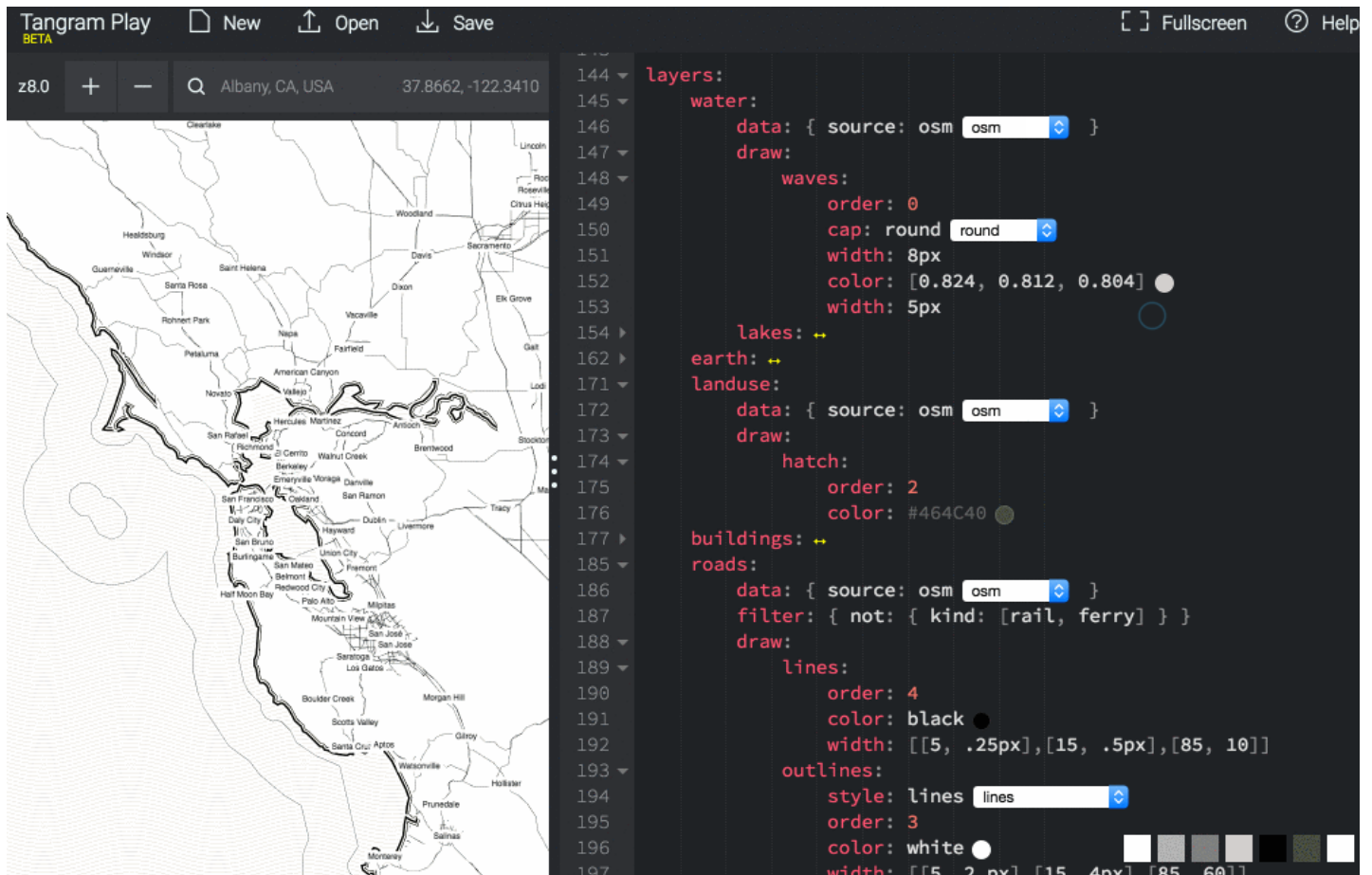
Map by my son

Looking for some map activities for your kids, at work or at home? Consider printing out these **Refill** (<http://tangrams.github.io/refill-style/#14/37.7912/-122.3952>) maps, grab some markers, and use them as coloring books!

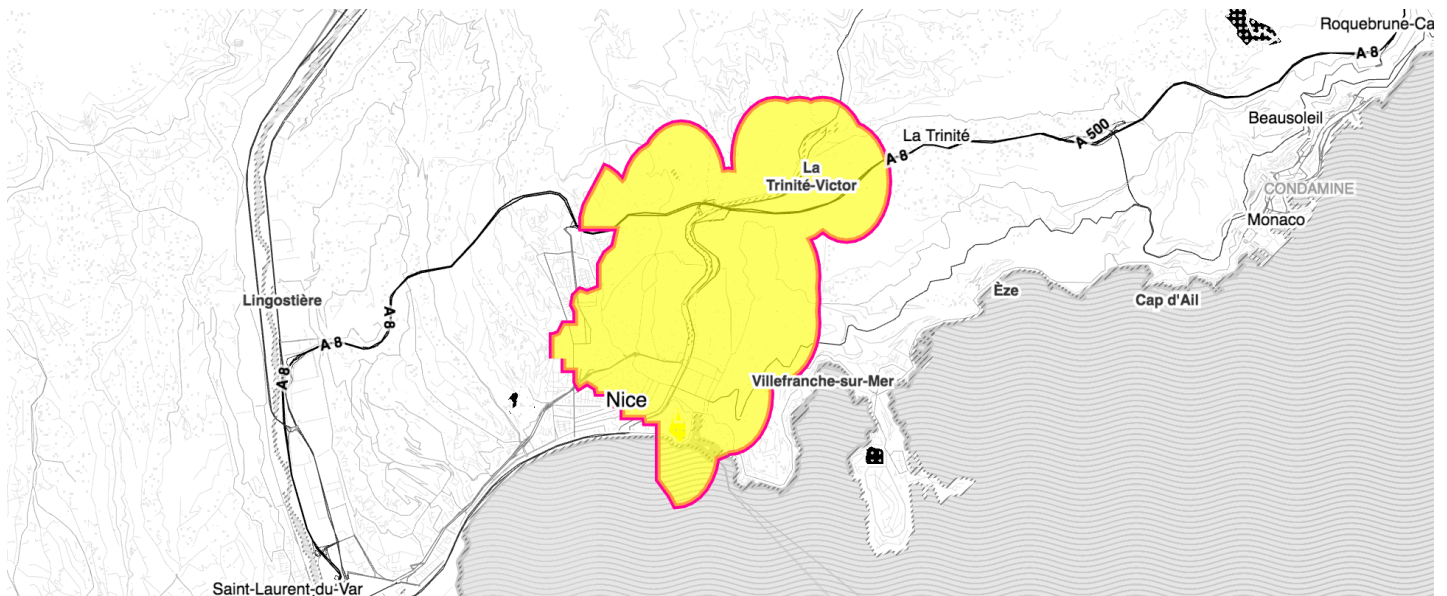


Or better yet, let them make their own map in **Tangram Play**

(<https://mapzen.com/tangram/play/?scene=https://cdn.rawgit.com/tangrams/tangram-sandbox/1d60a85ed384150d8a98c26fa30f5a4123c1224f/styles/press.yaml#8/38.018/-121.501>)! (The color pickers in the in styles make it easy to change things on maps.)



The **Who's on First spelunker** (<https://whosonfirst.mapzen.com/spelunker>) is a great way for kids to learn about places, borders, and hierarchies. Be sure to click on the **random place** (<https://whosonfirst.mapzen.com/spelunker/random/>) button!



Maps are great for kids every day, not just today. Let us know what your kids make!

· 28 April 2016 ·



John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

What a Relief: Global Test Pilots Wanted

[tangram \(/tag/tangram\)](#) [data \(/tag/data\)](#) [terrain \(/tag/terrain\)](#)

We're pleased to announce **worldwide elevation data** via a coverage upgrade to our terrain preview tiles. Additionally, in the United States **10 meter NED** data now augments the earlier NED 3 meter and 30 meter SRTM data for crisper, more consistent mountain detail.



Leaflet

(Open Scotland full screen ↗ (<https://tangrams.github.io/terrain-demos/?url=styles/normal-s-tiles.yaml#9/56.8107/-4.6678>))

Back in March we previewed tiles in California with Peter's **Mapping Mountains** (<https://mapzen.com/blog/mapping-mountains/>) post using a pre-release version of Tangram to generating hillshades client-side. That functionality has been finalized and is now available in Tangram **v0.7** (<https://github.com/tangrams/tangram/releases/tag/v0.7.0>) and Peter's demos have been **updated** (<https://github.com/tangrams/terrain-demos>).

Excited about tiled raw elevation data but want take it back to your desktop GIS or use it in the cloud? A new 512x512 **GeoTIFF** option joins our earlier web and mobile PNG format raw elevation Terrarium and processed Normal tiles. It goes well with Mapzen's **Metro Extract** (<https://mapzen.com/data/metro-extracts/>), nudge nudge. Tired of Mercator? **Skadi** format tiles offer raw elevation tiles in latlng projection.

Give our preview elevation tilesets a test flight in your own applications, no API key required.

Endpoints

Terrain tile endpoints are:

- <https://tile.mapzen.com/mapzen/terrain/v1/terrarium/{z}/{x}/{y}.png>
(<https://tile.mapzen.com/mapzen/terrain/v1/terrarium/%7Bz%7D/%7Bx%7D/%7By%7D.png>)
- <https://tile.mapzen.com/mapzen/terrain/v1/normal/{z}/{x}/{y}.png>
(<https://tile.mapzen.com/mapzen/terrain/v1/normal/%7Bz%7D/%7Bx%7D/%7By%7D.png>)
- <https://tile.mapzen.com/mapzen/terrain/v1/geotiff/{z}/{x}/{y}.tif>
(<https://tile.mapzen.com/mapzen/terrain/v1/geotiff/%7Bz%7D/%7Bx%7D/%7By%7D.tif>)
- <https://tile.mapzen.com/mapzen/terrain/v1/skadi/{N|S}{y}/{N|S}{y}{E|W}{x}.hgt.gz>
(<https://tile.mapzen.com/mapzen/terrain/v1/skadi/%7BN%7CS%7D%7By%7D/%7BN%7CS%7D%7By%7D%7BE%7CW%7D%7Bx%7D.hgt.gz>)

Tiles are available at zooms 0 through 14, with zoom 15 coverage filling out in the weeks to come.

(*) Note: GeoTIFF format tiles are 512x512 sized so request the parent tile's coordinate. For instance, if you're looking for a zoom 14 tile then request the parent tile at zoom 13.

Additional Amazon S3 Endpoints

If you're building in Amazon AWS we recommend using machines in the **us-east** region (the same region as the S3 bucket) and use the following endpoints for increased performance:

- <https://s3.amazonaws.com/elevation-tiles-prod/terrarium/{z}/{x}/{y}.png>
- <https://s3.amazonaws.com/elevation-tiles-prod/normal/{z}/{x}/{y}.png>
- <https://s3.amazonaws.com/elevation-tiles-prod/geotiff/{z}/{x}/{y}.tif>
- <https://s3.amazonaws.com/elevation-tiles-prod/skadi/{N|S}{y}/{N|S}{y}{E|W}{x}.hgt.gz>

File formats

- **Terrarium** format PNG tiles contain raw elevation data in meters, in Mercator projection (EPSG:3857). All values are positive with a 32,768 offset, split into the red, green, and blue channels, with 16 bits of integer and 8 bits of fraction. To decode:

```
(red * 256 + green + blue / 256) - 32768
```

- **Normal** format PNG tiles are processed elevation data with the the red, green, and blue values corresponding to the direction the pixel "surface" is facing (its XYZ vector), in Mercator projection (EPSG:3857). The alpha channel contains quantized elevation data with values suitable for common hypsometric tint ranges. High alpha channel values indicate lower elevation values (below sea level), making them more opaque.

Specifically, normal format alpha values are counted in (floored) elevation increments. Below sea level they start at -11,000 meters (Mariana Trench) and range to -1,000 meters in 1,000 meter increments, with more detail on the coastal shelf at -100, -50, -20, -10 and -1 meters and finally 0 (intertidal zone). Values above sea level are reported in 20 meter increments to 3,000 meters, then 50 meter increments until 6,000 meters, and then 100 meter increments until 8,900 meters (Mount Everest).

- **GeoTIFF** format tiles are raw elevation data suitable for analytical use and are optimized to reduce transfer costs in 512x512 tile sizes but with internal 256x256 image pyramiding, in Mercator projection (EPSG:3857). Allow for the larger tile size by referring to the tile coordinate of {z-1} parent tile.
- **Skadi** format tiles are raw elevation data in unprojected latlng (EPSG:4326) 1°x1° tiles, used by the Mapzen Elevation Service. Essentially they are just the SRTMGL1 format tiles but with global coverage. See **the SRTM guide** (https://lpdaac.usgs.gov/sites/default/files/public/measures/docs/NASA_SRTM_V3.pdf) for exact format specifications.

Data sources

Mapzen aggregates elevation data from several open data providers including 3 meter and 10 meter **NED** (<http://nationalmap.gov/elevation.html>) in the United States, 30 meter **SRTM** (<https://lta.cr.usgs.gov/SRTM>) globally, and coarser **GMTED** (http://topotools.cr.usgs.gov/gmted_viewer/) zoomed out and **ETOPO1** (<https://www.ngdc.noaa.gov/mgg/global/global.html>) to fill in bathymetry, all powered by GDAL/OGR **VRTs** (http://www.gdal.org/drv_vrt.html) and some special sauce.

Could we provide better coverage in your area? Please recommend additional open datasets to include by **filing an issue** (<http://github.com/mapzen/joerd/issues/new>).

Related service

To query elevation at a single point or along a path almost anywhere in the globe, **sign up for an API key** (<https://mapzen.com/developers/>) and check out the **Mapzen Elevation Service** (<https://mapzen.com/documentation/elevation/>).

Feedback

Have any feedback on the tiles? We're all ears during this extended "beta" preview of elevation tiles. Please direct feedback to **Nathaniel Vaughn Kelso** (<https://github.com/nvkelso>), our Head of Data & Cartography, at **hello@mapzen.com** (<mailto:hello@mapzen.com>)!

CORRECTION: This post was updated on 2016.05.03 to detail additional S3 endpoints and clarify GeoTIFF file extension as `.tif` with one f instead of two.

UPDATE Jan 19, 2017: We've also deprecated the terrain-preview beta endpoints and have updated this blog with the proper URLs_

· 03 May 2016 ·



Nathaniel Vaughn Kelso

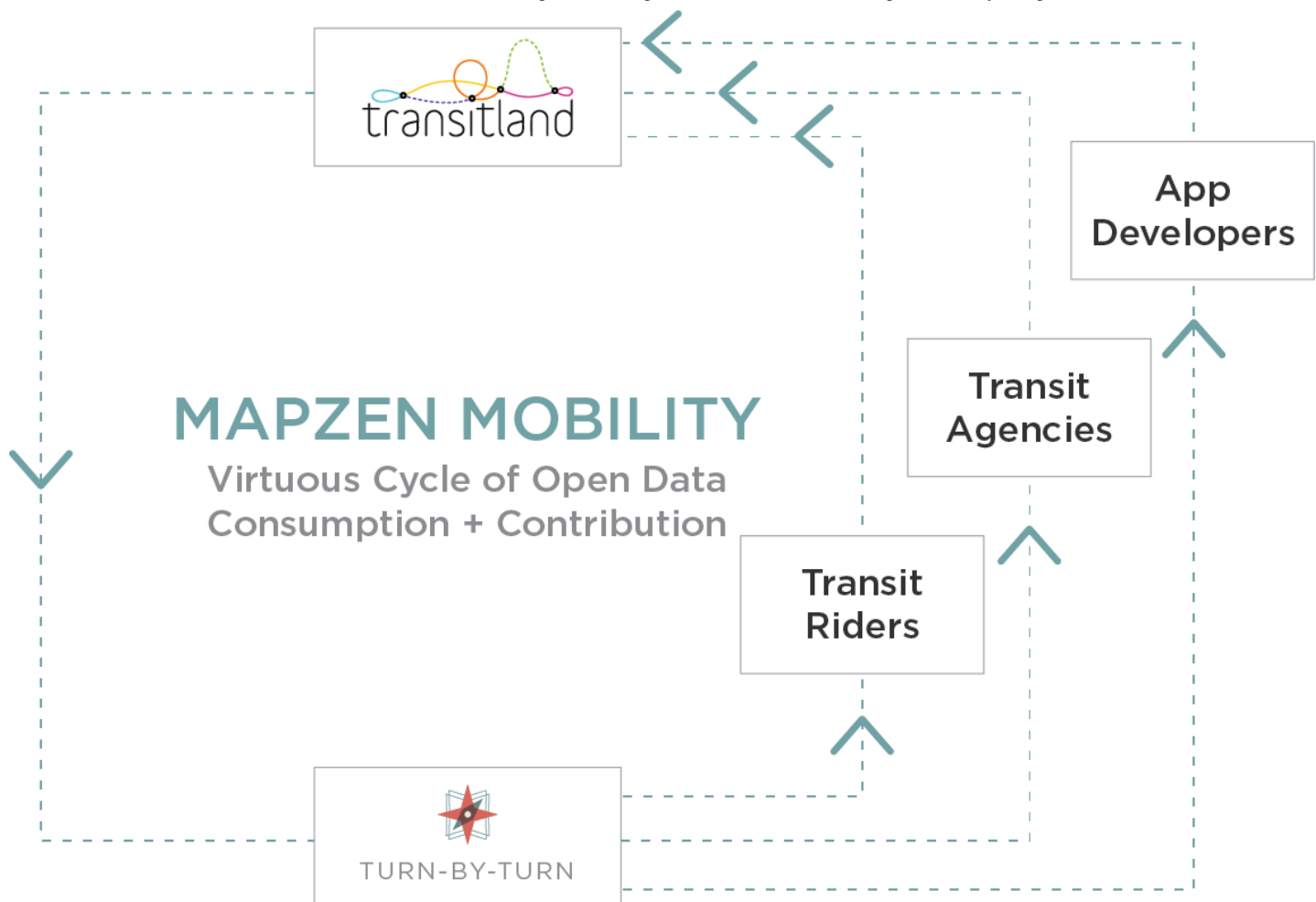
Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

© 2017 Mapzen

The new (multimodal, open-source, open-data) rules of the road: Mapzen Mobility

mobility (/tag/mobility) **routing** (/tag/routing) **transitland** (/tag/transitland)

Steamboats plying rivers, horses and buggies trundling down dirt roads, cars speeding along blacktop highways. Transportation is often told as a story of increasing speed and range, each leap forward enabled by a breakthrough in vehicle propulsion or civil engineering. The way we get around continues to evolve. Our commutes to work and school, the way we meet up with family and friends, and how we shop are now shaped by a variety of transportation technologies—car-sharing, ride-sharing, bike-sharing, public-transit, car-ownership, or same-day delivery are only some of the options in many large metro areas. This is the new nature of progress: multiple modes of transportation, some owned privately and others operated publicly, some offered as a service and others sold as a package, all available to be stitched together to best fit the unique needs of each individual. ***To hasten this twenty-first century form of access, we're creating Mapzen Mobility, a new line of products that enable multimodal transportation through a "virtuous cycle" of open data consumption and contribution.***



An open-source toolkit for multimodal transportation

Under this umbrella of Mapzen Mobility, we're offering a toolkit of services, powered exclusively by open-source software and open data:

Turn-by-Turn

Mapzen Turn-by-Turn (</projects/turn-by-turn>) guides you from A to B by car, bike, foot, and—as of today—multimodal combinations involving walking and riding public transit. **Today's release makes Valhalla, the open-source project that powers Mapzen Turn-by-Turn, the first open-source routing engine able to serve transit directions across multiple metropolitan regions** (</blog/transit-routing/>).

Apps can now use Mapzen Turn-by-Turn to plan multimodal journeys, create narratives to guide users by text and by voice, and update those journeys on the fly from mobile devices. Mapzen Turn-by-Turn draws data from OpenStreetMap and from Transitland, the open transit data aggregation project that Mapzen sponsors. Apps can also query Transitland's API to build maps and analyses that enrich that journey and provide context around Points A and B, as well as the

many multimodal transportation options that connect them. Journeys planned by Mapzen Turn-by-Turn and data in Transitland all include Onestop IDs, an open identifier scheme that catalogs transit operators, stops, and routes from around the world.

Read more about **how Mapzen Turn-by-Turn multimodal routing works (/blog/transit-routing)**. Try a **demo of multimodal routing in San Francisco (/projects/turn-by-turn)**. And when you're ready to try the underlying API with coverage for the San Francisco Bay Area, New York City, and Rome (<https://transit.land/news/2016/03/24/transitland-in-italy.html>) sign up for a **Turn-by-Turn developer key (/developers)**. Please stay tuned for updates as we expand **Turn-by-Turn's transit coverage around the world (https://transit.land/feed-registry/?import_level=4)** and its multimodal combinations even further.

Transitland

Transitland (https://transit.land) is a community project sponsored by Mapzen to aggregate and improve transit data from urban and rural areas around the world. Earlier this year, Transitland opened to contributions of transit data feeds, and has steadily grown to cover public-transit networks from a gondola in Portland, Oregon, to buses and subways in Rome, to the informal network of *matatus* in Nairobi, Kenya. It's this collection of stop, route, and schedule data that powers Mapzen Turn-by-Turn. But it's also available to other consumers and contributors. We welcome everyone to use Transitland data to power their own routing engines.

Do you also work with transit data? Let's collaborate on collecting and improving transit data in the open. That should be shared infrastructure. We can compete on the services, like routing engines and apps, that each of our organizations build on top of that common set of data.

To see Transitland's expanding coverage browse the **Feed Registry (https://transit.land/feed-registry)** and follow **Transitland's news and updates (https://transit.land/news)**.

Mapzen Matrix

Trying to run more than one errand in the day, or trying to start your own delivery service? **Mapzen Matrix (/blog/matrix)** can compute the times and distances between up to 50 different origins and destinations at once. Like Mapzen Turn-by-Turn, it's powered by OpenStreetMap and works anywhere around the world. **Try a demo of Mapzen Matrix in New York City (http://valhalla.github.io/demos/matrix/)** and **sign up for a developer API key to use Mapzen Matrix around the world (/developers)**.

Mapzen Elevation

Want to avoid hills while running those errands, or looking to optimize your cycling workouts? **Mapzen Elevation (/blog/moving-on-up)** can be paired with Mapzen Turn-by-Turn to plan your journeys with elevation gain in mind. **Try a demo of Elevation in the Alps (<https://valhalla.github.io/demos/elevation/>)** and **sign up for a developer API key to use Mapzen Elevation around the world (/developers)**.

The new rules of the (multimodal, open) road

From these four initial services in the Mapzen Mobility toolkit, our “rules of the road” are clear:

- *All modes of transportation are created equal.* We’re not a car company. We think walking, cycling, and car-sharing are just as important as private car ownership and operation. Mapzen Mobility supports all modes of transportation, and in the future, our services will support any possible combination of modes on a journey.
- *Open data stitches together public and private transportation networks.* Fixed-route buses and subways are usually operated by public agencies (at least here in the U.S.), while ride-sharing networks are owned by private companies and operated by a mix of private contractors. A trip that involves many modes will likely traverse both publicly operated and privately owned transportation systems. Mapzen Mobility uses only open data, since it’s the common denominator between public and private organizations. Rather than hoarding or selling raw data, Mapzen believes that true value comes from services, applications, and experiences built on top of mapping, scheduling, usage, and ticketing data.
- *People first.* In technical terms, what makes twenty-first century mobility novel is that it combines transportation technology with information and communication technology. Many of these new on-demand services are possible only because of smartphones and wireless data networks. Thinking more broadly, these advances in multimodal transportation are refocusing attention from infrastructure and software to the lives and needs of people. *Where do you want to go? Where do you need to go? How can you and your neighbors access places of work, learning, and health?* Mapzen Mobility is here to support the freedom of movement, with everyday people in mind.

Join us for the ride

Now that you’ve heard our vision for multimodal mobility powered exclusively by open data, try **Mapzen Turn-by-Turn (/projects/turn-by-turn)**. As of today, it’s the world’s first open-source, global, multimodal routing engine. We’ll be adding coverage and functionality to Turn-by-Turn

service (and the Valhalla open-source project) over the next few months. Also, watch for more additions to the Mapzen Mobility product line.

We welcome your comments, questions, and ideas about Mapzen Mobility at **hello@mapzen.com** (<mailto:hello@mapzen.com>) or **@mapzen** (<https://twitter.com/mapzen>). Together we're going places.

· 03 May 2016 ·



Drew Dara-Abrams

Drew leads Mapzen Mobility products (and aspires to being a flâneur).



David Nesbitt

Dave leads Mapzen Mobility engineering. Rides a variety of 2 wheel vehicles.

© 2017 Mapzen

Let's take a ~~road trip~~ transit journey to Valhalla, NY using Mapzen Turn-by-Turn and Transitland

mobility (/tag/mobility) **routing** (/tag/routing) **transitland** (/tag/transitland)



Photos by Heidi Knisely

Back in February, **Transitland** (<https://transit.land>) opened up to **contributions of transit data from around the world** (</blog/help-us-catalog-the-transit-feeds-of-the-world>). Today we are ready announce that **Mapzen Turn-by-Turn** (</projects/turn-by-turn>) now uses data

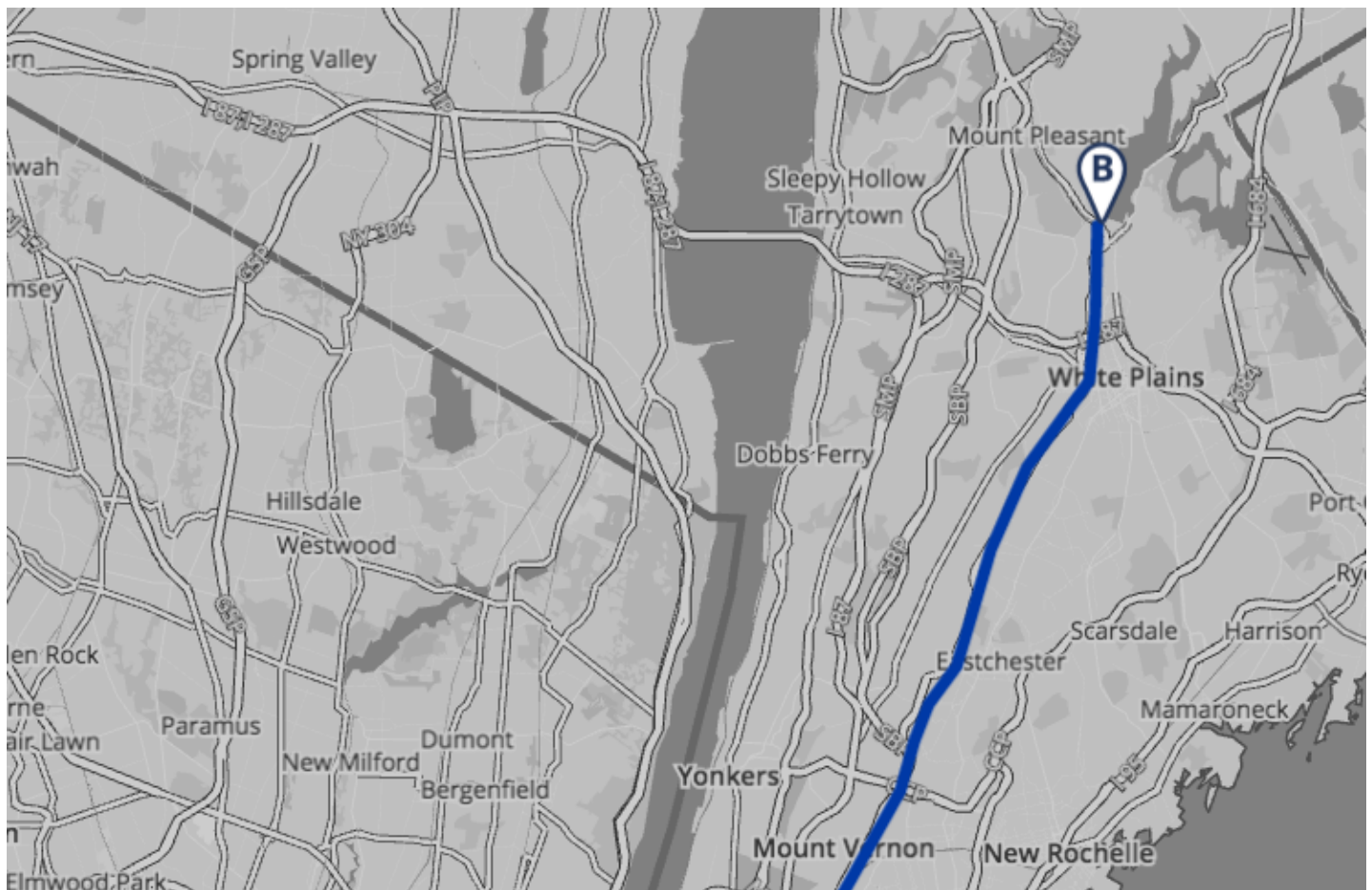
from Transitland to plan journeys on subways, buses, trains, and ferries in San Francisco and New York City. Together, Transitland and Mapzen Turn-by-Turn form the foundation of **Mapzen Mobility (/blog/introducing-mapzen-mobility)**, our new initiative to improve multimodal transportation through open data.

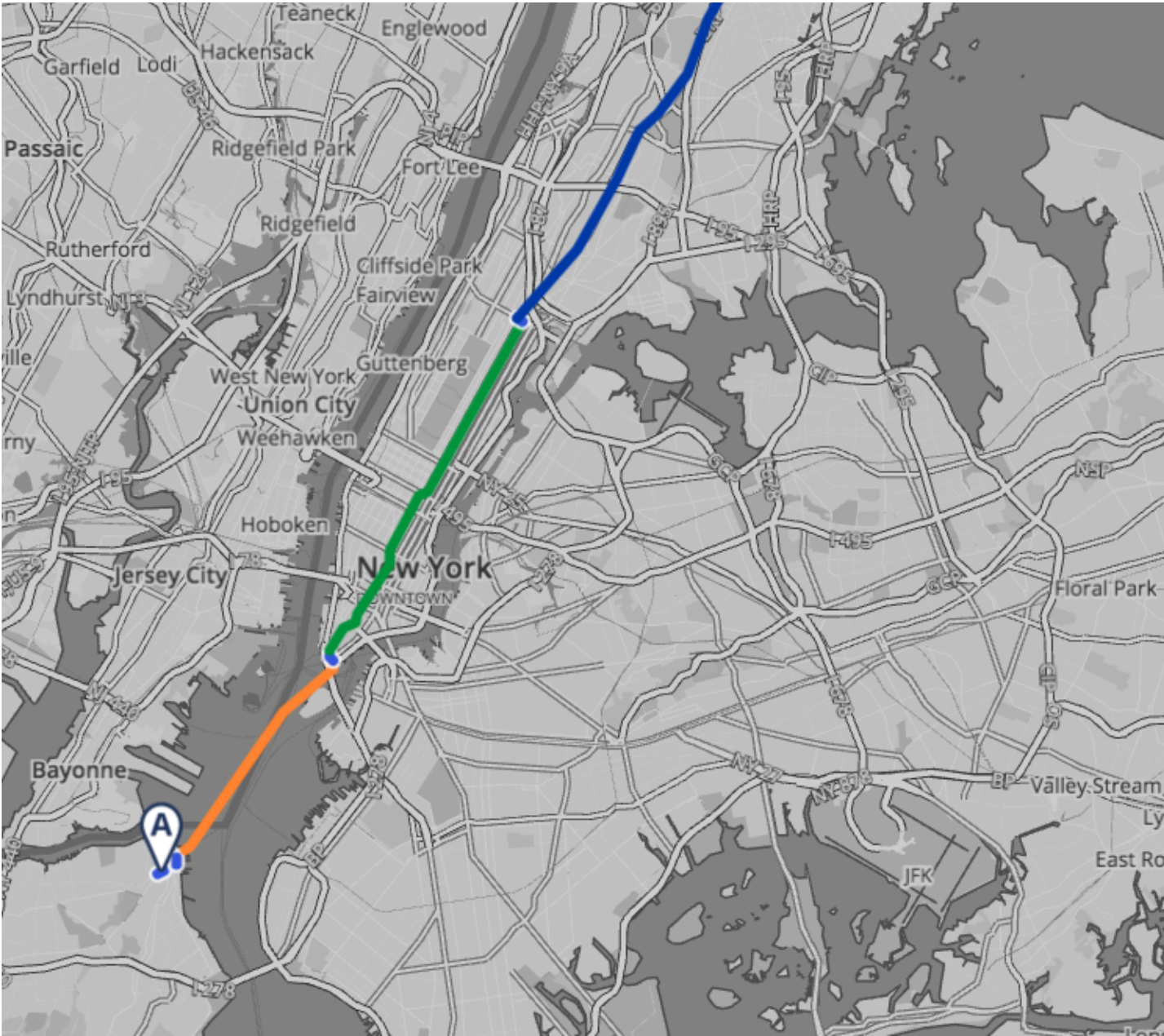
“Multimodal” refers to routing via walking and public transit in **Mapzen Turn-by-Turn (/projects/turn-by-turn)**, using time-based schedules of the public transportation. We can walk to a station, utilize public transportation, disembark and walk to our final destination. By default, we try to use rail before buses and attempt to avoid transfers. However, these values can all be tweaked to your liking. **Sign up for an API key (<https://mapzen.com/developers>) and dig into our documentation (/documentation/turn-by-turn/api-reference/) to learn about the API request format.**

But first let's take a look and see what we can do! Going from Staten Island to Valhalla, NY is a long but fun trip **since it contains ferry, subway, and rail maneuvers**




(<https://mapzen.com/projects/turn-by-turn/?>









d=4&lat=40.7259&lng=-73.9805&z=12&c=multimodal&st_lat=40.63688312646408&st_lng=-74.08287048339844&st=64%20Brook%20Street%2C%20Staten%20Island%2C%20NY&end_lat=41.07262208878544&end_lng=-73.77233505249023&end=2%20Cleveland%20Street%2C%20Mount%20Pleasant%2C%20NY&dt=&dt_type=).





In the narration, we provide written departure and arrival time instructions for each transit portion of our trip.

	Enter the St. George Ferry Terminal station.	17 m
	<i>Depart: 9:30 PM from St. George Ferry Terminal.</i> Take the Staten Island Ferry. (1 stop) <i>Arrive: 9:55 PM at Whitehall Ferry Terminal.</i>	8.111 km
	Exit the Whitehall Ferry Terminal station.	30 m

	<i>collapsing walking maneuvers</i>	
	Enter the Bowling Green station.	22 m
	<i>Depart: 10:04 PM from Bowling Green.</i> Take the 4 toward WOODLAWN. (8 stops) <i>Arrive: 10:26 PM at 125 St.</i>	12.86 km
	Exit the 125 St station.	31 m
	Walk northwest on East 125th Street.	142 m
	Enter the Harlem-125th St. station.	29 m
	<i>Depart: 10:32 PM from Harlem-125th St.</i> Take the Harlem toward Southeast. (3 stops) <i>Arrive: 11:03 PM at Valhalla.</i>	33.316 km
	Exit the Valhalla station.	28 m

(Normally the walking portions of the route would be included, but we're focusing on the transit maneuvers for now.)

Use Options

As stated earlier, we can tweak our API defaults (`use_bus` , `use_rail` , and `use_transfers`). The defaults are `use_bus = 0.3` , `use_rail = 0.6` , and `use_transfers = 0.3` .

- `use_bus` User's desire to use buses. Range of values from 0 (try to avoid buses) to 1 (strong preference for riding buses).
- `use_rail` User's desire to use rail/subway/metro. Range of values from 0 (try to avoid rail) to 1 (strong preference for riding rail).
- `use_transfers` User's desire to favor transfers. Range of values from 0 (try to avoid transfers) to 1 (totally comfortable with transfers).

Note that these parameters can be changed in the URL in the **Mapzen Turn-by-Turn Transit demo** (<https://mapzen.com/projects/turn-by-turn>).

Date and Time Options







For multimodal routes, we default to the current time, though we can specify a departure date and time. (Note that this is in the timezone of the origin location.)

date_time

- type
 - 0 — Current depart time
 - 1 — Specified depart time
- value
 - The date and time to depart. ISO 8601 format (YYYY-MM-DDThh:mm). For example "2016-03-29T08:06"

Say you are in San Francisco around 4th & Townsend and want to go to Beach Street by Fishermans Wharf. Also, you want to depart on March 29 at 6:30 AM (`2016-03-29T06:30`) favoring buses; `use_bus` is now set at 0.5 and `use_rail` set at 0.3.

Notice in the results that we take the bus in the morning at 6:35 AM and avoid the rail lines.

	Walk southwest on Berry Street.	116 m
	Turn right onto 4th Street/Fourth Street.	210 m
	Turn right onto Townsend Street.	25 m
	<p><i>Depart: 6:35 AM from TOWNSEND ST & 4TH ST.</i></p> <p>Take the 30 toward the Marina District. (18 stops)</p> <p><i>Arrive: 6:56 AM at Columbus Ave & North Point St.</i></p>	4.158 km
	Walk northwest on Columbus Avenue.	55 m
	Turn right onto Leavenworth Street.	184 m



You have arrived at your destination.

How did we do it...in a nutshell:

Currently, once a week **Turn by Turn's** (</projects/turn-by-turn>) `transit_fetcher` grabs transit routes, stops, **schedule stop pairs (departures)** (</blog/the-transit-dimension-transit-land-schedule-api>), operators, and route stop patterns (shapes) using the **Transitland Datastore API** (<https://transit.land/how-it-works/#slide-3>). It saves the data to temporary transit data tiles in protocol buffer format. During the normal data update that creates routing graph tiles, a `transit_builder` process creates transit nodes and edges that are connected to nodes in the existing graph. **Transitland** (<https://transit.land/>) associates transit stops to OpenStreetMap ways using Mapzen Turn-by-Turn's `locate` method. This is done whenever transit stops are created or updated in the Transitland Datastore. In addition to modifying the routing graph, transit departures, routes, and stop information are stored within the routing tiles.

When you run a multimodal route at a specific date and time and encounter a transit node that contains schedule information, we try to obtain a departure from that node at the date and time you encountered that particular node. If a departure is found, we will “walk” the transit edges and nodes applying time costs and penalties along the way for transfers. This is repeated until the best, multimodal path is returned.

Coming soon to a city near you

Mapzen's motto is “start where you are”, so if you see ways to improve our multimodal routing please let us know. Also, have a look at our **documentation** (</documentation/turn-by-turn/api-reference/>) to see the structure of the API request and response and **sign up for a free developer key to start using Mapzen Turn-by-Turn** (</developers>).

If you're not in the SF Bay Area, the NYC metro region, or Rome, we hope to reach your part of the world with Mapzen Turn-by-Turn routing soon. Please help expand Transitland's coverage by **contributing feeds to the Feed Registry** (<https://transit.land/news/2016/02/19/get-started-add-feeds.html>). In the coming months, we'll be ingesting even more of these feeds into Turn-by-Turn. To see which transit operators are routable, follow **this link to a filtered view of the Feed Registry** (https://transit.land/feed-registry/?import_level=4). As always, should you have any questions, write to us at hello@mapzen.com (<mailto:hello@mapzen.com>).

· 03 May 2016 ·



Greg Knisely

Greg is a data munger and software engineer for Mapzen's routing software.

© 2017 Mapzen

Office hours at Mapzen

Come in, Mapzen is open!



We're inviting our users to join us for open Office Hours. We'll have staff available from our engineering and developer support teams to provide personalized technical assistance and training—whether you are new to the Mapzen platform or have been using our tools for a while.

Mapzen will be hosting these events regularly, starting in our San Francisco office on Wednesday, May 18, from 9 to 11 am. Office Hours in other cities to follow this summer.

This is a great time to:

- enjoy some tasty breakfast foods
- show off what you're working on
- troubleshoot issues you are having with our tools
- let us know how we can improve our tools or documentation
- learn how Mapzen can fit into your project's workflow
- say hello and meet our teams

Sign up (<http://goo.gl/forms/6PjFyGTww0>) to let us know you're coming. If you give us an idea about what you want to discuss, we can make sure we have the right team here to help you.

Join us:

- Mapzen - San Francisco office (closest transit: Embarcadero station)
- Wednesday, May 18, 2016
- 9 to 11 am
- RSVP: <http://goo.gl/forms/6PjFyGTww0> (<http://goo.gl/forms/6PjFyGTww0>)

Of course, if you need help at any time, or want us to hold office hours in your city, send us a note at hello@mapzen.com (<mailto:hello@mapzen.com>)!

· 05 May 2016 ·



Rhonda Glennon

Rhonda is Mapzen's technical publications manager and writes about maps and developer tools.

© 2017 Mapzen

Vector Tiles v0.10 -- get yourself outdoors and on a bike

vector-tiles (</tag/vector-tiles>) **data** (</tag/data>) **documentation**
(</tag/documentation>)

Mapzen has released **v0.10** (<https://github.com/mapzen/vector-datasource/releases/tag/v0.10.0>) of our Vector Tile service. This version enables outdoor recreation maps and bicycling maps, and reorganizes a few labels.

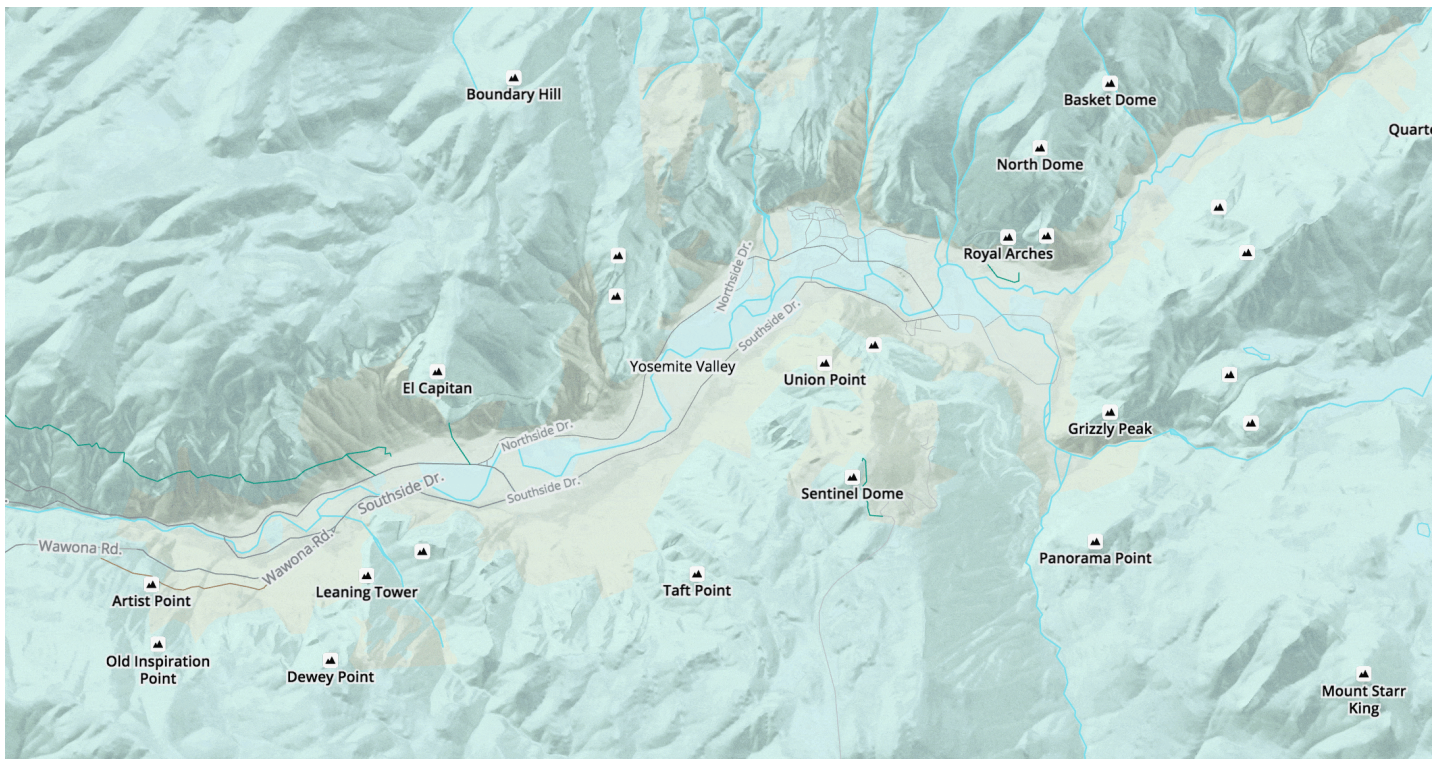
Head outside

Just in time for spring, the v0.10 release enables you to head to the nearest trailhead and get outside!

OpenStreetMap includes amazing detail for hiking paths, caravan sites, picnic sites, fire pits, BBQs, hot springs, geysers, rapids, waterfalls, and islands, but it can be a challenge to pull these out and get them onto a map. These and other features are now easier to access in **v0.10** (<https://github.com/mapzen/vector-datasource/releases/tag/v0.10.0>), so zoom in and explore richer coverage of place data in parks, water parks, beach resorts, and summer camps.

Wondering where that trail is? We now show nationally and regionally important trails earlier zoomed out to 11. Curious how high that mountain is? We now include elevation on peak and volcano features, and waterfall heights.

Here's a quick preview comparing the level of detail at zoom level 13 in the old v0.9 tiles vs the new v0.10 vector tiles.



(https://mapzen-assets.s3.amazonaws.com/images/tiles-0.10/yosemite_v0.9_v0.10_outdoor.gif)

Click to zoom (https://mapzen-assets.s3.amazonaws.com/images/tiles-0.10/yosemite_v0.9_v0.10_outdoor.gif)!

Bike across town and around the world

Celebrating May as **Bike Month** (<http://bikeleague.org/bikemonth>), Mapzen Vector Tiles v0.10 includes additional information to highlight bike lanes (known as `cycleways` in OpenStreetMap), bike paths, routes, and junctions.

We've also revised how we treat bike stores, bike rental shops, bike rental stations. We indicate nationally and regionally important bike routes out to zoom 11.



21 European bike-only traffic signs, images **via Wikipedia**

(https://en.wikipedia.org/wiki/Comparison_of_European_road_signs), GIF by Burrito Justice

Want to add bicycle information to OpenStreetMap in your town? Watch the Mapzen blog for an upcoming **Targeted Editing** (<https://mapzen.com/tag/targeted-editing/>) post.

Label changes

We've moved a few features around to make them easier to find. But you'll need to update your stylesheets to restore these features to your map.

Continent label placements are now in the earth layer (they were in the places layer). Ocean and sea label placements are similarly moved to the water layer. We've normalized kind values at low zooms to be more consistent with those at high zooms in the boundaries and water layers.

Check out the full story in the **v0.10 release notes** (<https://github.com/mapzen/vector-datasource/releases/tag/v0.10.0>).

As we work towards a v1.0 release this summer we'd love to hear your feedback on Mapzen Vector Tiles. Please say **hello@mapzen.com** (<mailto:hello@mapzen.com>)! And **sign up for an API key** (<https://mapzen.com/developers>) and get even better bike and outdoor vectors into your maps!

· 11 May 2016 ·

**Nathaniel Vaughn Kelso**

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

**Rob Marianski**

Software engineer working on cutting tiles and infrastructure. Functional programming enthusiast.

**Matt Amos**

OpenStreetMap developer-contributor and chilli enthusiast. Writes vector tile and other data-mastication tools at Mapzen.

© 2017 Mapzen

Prime Server ZMQ

engineering (/tag/engineering) routing (/tag/routing)

```

0
, . . . . .
| ) | | | | (. - ' \ - ( - ' | \ / ( - ' |
| - ' - ' - ' - ' - ' - ' - ' - ' - ' - ' - '
|
|
|

```

The Introduction

`prime_server` is an API for building HTTP SOAs. Less the acronyms: it's a library and executables which marry an http server to talk to clients, with a distributed computing backend to do the work. As an example we've made a sample application that will tell you if a given number is prime. If you'd like to give that a shot try installing and running it:

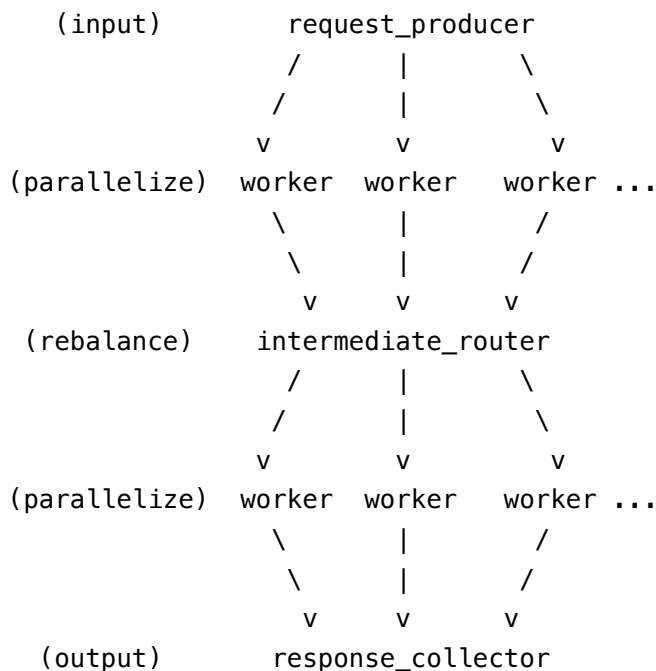
```

sudo add-apt-repository ppa:kevinkreiser/prime-server
sudo apt-get update
sudo apt-get install libprime-server0 libprime-server-dev prime-server-bin
prime_serverd tcp://*:8002 &
curl "http://localhost:8002/is_prime?possible_prime=32416190071"
killall prime_serverd

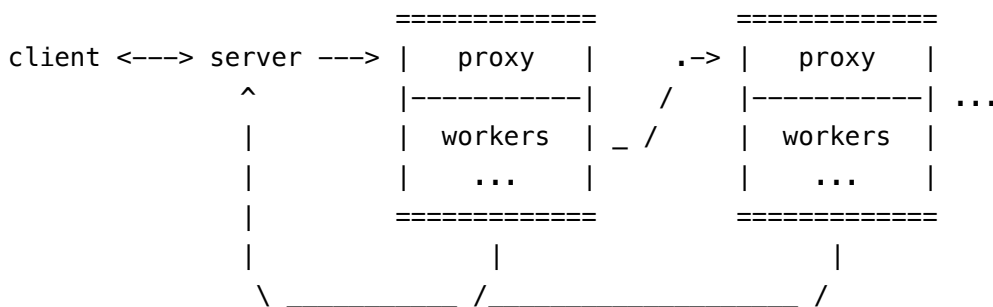
```

The Point

We wanted to make a tool that lets you build a system that is pipelined and parallelized, i.e. the ZMQ “butterfly” or “parallel pipeline” pattern. See **this tutorial** (<http://zeromq.org/tutorials:butterfly>). We'll get to why in a bit but first, this is what it should look like:



This seems like a pretty good pattern for some offline scientific code that pumps jobs into the system and waits for the results to land at the bottom. However this isn't very useful for online systems that face users. For that we need some kind of loopback so we can get the result back to the requester. Something like:



A client (a browser or just a separate thread) makes a request to a server. The server listens for new requests and replies when the backend parts send back results. The backend is comprised of load balancing proxies between layers of worker pools. In real life you may run these in different processes or on different machines. We use threads in the example server to conveniently simulate this within a single process, so **please note** (https://github.com/kevinkreiser/prime_server/blob/master/src/prime_serverd.cpp) the lack of any mutex/locking patterns (thank you ZMQ!).

This system lets you handle requests that can be decomposed into a single pipeline of multiple steps. This is useful when certain steps take longer than others since you can scale them independently of one another. It doesn't really handle requests which would need multiple pipelines, or branching pipelines (unless workers can do more than one job). To fix this we could upgrade the workers to be able to forward work to more than one proxy. This would allow heterogeneous workflows without having to make smarter/larger pluripotent workers and therefore would allow scaling of various workflows independently of each other. This sounds great, but the configuration would be a mess. An easier approach would be just to run a separate cluster per workflow, but there are pros and cons there too. We'll talk more about options here later.

You may be asking yourself, *why on earth are all of the worker pools hooked into the loopback?* This has two important functions:

1. A request could enter an error state at any stage of the pipeline. It's important to be able to signal this back to the client as soon as possible.
2. More generally, certain requests have known results (error or otherwise) without going through all stages of the pipeline.

The Impetus

The toy example of an HTTP service that computes whether or not a number is prime is a simple illustration of why someone might want a setup as described above. Is 3 prime? Yes. Is 6 prime? No. Is 32416190071 prime? Give me a while.. But it's not the actual use-case that drove the creation of this project. Having worked on a few service oriented architectures, we noticed that we'd compiled what amounted to a wishlist of architectural features. In buzz-word form those were roughly:

- Simplicity
- Flexibility
- Fault Tolerance
- Separation of Concerns
- Throughput
- Load Balancing
- Fair Queuing
- Quality of Service
- Non-blocking
- Web Scale (just kidding)

We needed to handle HTTP requests that would have widely varying degrees of complexity. Specifically, we were writing **some software** (<https://github.com/valhalla>) that does shortest (for some value of short) path computations over large graphs. For example, users would be able to make requests to get the best route by car/foot/bike/etc. from London to Edinburgh, which may take 10s of milliseconds. In contrast, a user would also be able to ask for the route from Capetown to Beijing, which could take *thousands* of milliseconds, i.e. several seconds. Different requests can vary in computation time by orders of magnitude.

But its worse than that! You might imagine that finding a path through a graph sounds pretty straight forward, but making that path *useful* to the client requires a bunch of extra work. Conveniently, or sometimes through great effort, decomposing a problem into discrete steps can help you with that wish list. Doing so in the context of an HTTP server API requires a little extra consideration. Basically, catering to (or hoping for) many simultaneous users makes most of those buzz-words relevant. Especially the last one (again just kidding).

The Path

This was so fun to build for so many reasons. The first thing to do was prove out the ZMQ butterfly pattern as a tiny Github gist. The idea was that we could put an HTTP server in front of this pattern and hit some of those pesky wishlist items just by having separate stages of the pipeline. Surprisingly, learning this pattern and figuring out how it would work in a concrete scenario was another delight.

Public Service Announcement: Have you read the ZeroMQ **documentation** (<http://zguide.zeromq.org/page:all>)? We're not usually huge fans of technical writing, but it is an absolute joy to read. **Warning:** it may cause you to re-think many many past code design decisions. Don't worry though, that feeling of embarrassment over past poor decisions is just an indicator of making better informed ones in the future! Massive kudos to Pieter Hintjens who wrote that amazing document. If you are hungry for more good writing checkout his **blog** (<http://hintjens.com/>) or that of another ZMQ author **Martin Sústrik** (<http://250bpm.com/>). Both are fantastic reading!

Back to the gist. With so little setup around writing, running and debugging your code you can throw it away more easily if it doesn't work out. So we did that. Again and again, until we were left with what we came for; a parallelized pipeline whose stages had a loopback to a common entryptoint.

Next we scoured the internet for HTTP servers that had ZMQ bindings to put in front of our pipeline. As is no surprise, there are lots of good options out there! We spent some time prototyping using a few different ones but then we stumbled upon a very interesting search result. It was a **blog post (<http://hintjens.com/blog:42>)** from Pieter Hintjens about the ZMQ_STREAM socket type. It describes, with examples, how the socket works and ends up making a small web server using it. Hintjens goes on to say that a lot more work would be needed for a full fledged HTTP server and describes some of the missing parts in a bit more detail.

The idea was enticing; could we build a minimal HTTP server with just ZMQ to sit in front of our pipeline? We threw some stuff into a gist once again and started testing. Before long, and with very little code, we had something! From there though, it was on to writing an HTTP state machine to handle the streaming nature of the socket type. Writing state machines, especially against a couple of protocol versions at the same time is torture in terms code re-use. But we'll get to that in future work section.

The API

The API consists of essentially 3 parts:

- Client/server stuff - the bits that make and answer requests. The server stands between clients and the pipeline of workers and load balancing proxies.
- Proxy/worker stuff - the bits that fulfill the requests. The proxy sits between a pool of workers at a given stage in the pipeline and the next stage. The proxy knows what workers are available to do work and will not send on a request until a worker is available to take it. The workers are responsible to either send their results to another stage of the pipeline (the next proxy) or a sensible, protocol specific, response back to the server who will forward it on to the client.
- Protocol stuff - the bits that parse and serialize requests and responses respectively. It's a prearranged format that makes it possible for the client to speak something that the worker understands and vice-versa. HTTP is pretty useful here but other protocols exist. You can even create your own if you like. Also note that intermediate stages of workers can talk whatever protocol they like to each other.

So you want to make a web service. How can you do that... For the sake of example, let's say you want to do that all in the same process... In real life (i.e. production) you don't want to do that because, well, the wishlist again. But yeah let's just learn the easy way shall we?

Well what do you want to build? “Let’s make the ‘**B**eautiful **U**nicode **T**ext **T**ransmission’ service, the only API featuring artisanal text art!”, you say. We’re not sure we like where you’re going with this but hey, it’s your service! Let’s get the library installed:

```
sudo add-apt-repository ppa:kevinkreiser/prime-server
sudo apt-get update
sudo apt-get install libprime-server-dev
```

Ok great, let’s write our program against it, call it `art.cpp` . We’ll start by including a few things we’ll need:

```
//prime_server guts
#include <prime_server/prime_server.hpp>
#include <prime_server/http_protocol.hpp>
using namespace prime_server;

//nuts and bolts required
#include <thread>
#include <functional>
#include <chrono>
#include <string>
#include <list>
#include <vector>
#include <csignal>

//configuration constants for various sockets
const std::string server_endpoint = "tcp://*:8002";
const std::string result_endpoint = "ipc:///tmp/result_endpoint";
const std::string proxy_endpoint = "ipc:///tmp/proxy_endpoint";

//assortment of artisional content
const std::vector<std::string> art = { "(_,_)", "(_|_)", "(_*_)",
                                       "(~x~)", "(~i~)", "(~ε~)" };
```

So first off we’re including a couple of bits from `libprime_server` itself mainly for the setup of the pipeline and, as you guessed it, the stuff we need to talk the HTTP protocol. After that we include a bunch of standard data structures that we’ll make use of throughout the program, it’ll be pretty obvious.. Then we have some configuration. Basically we have to tell the different threads’ sockets where to find each other. The first one there is a `tcp` socket so that webclients can connect to us through normal means. The other two are unix domain sockets but could be `tcp` if you want to run different parts of this on different machines. For example you

could run one stage of the pipeline on machines with fat graphics cards for the GPGPU win, whereas another stage might run better on machines with metric tons of RAM. Finally we have our super sweet text art. Alright now how do we actually return a response to someone looking for some 'Textual Unicode Stuff of High Intellectual Excitement'?

```
//actually serve up content
worker_t::result_t art_work(const std::list<zmq::message_t>& job, void* request_info) {
    //false means this is going back to the client, there is no next stage of the pipeline
    worker_t::result_t result{false};
    //this type differs per protocol hence the void* fun
    auto& info = *static_cast<http_request_t::info_t*>(request_info);
    http_response_t response;
    try {
        //TODO: actually use/validate the request parameters
        auto request = http_request_t::from_string(
            static_cast<const char*>(job.front().data()), job.front().size());
        //get your art here
        response = http_response_t(200, "OK", art[info.id % art.size()]);
    }
    catch(const std::exception& e) {
        //complain
        response = http_response_t(400, "Bad Request", e.what());
    }
    //does some tricky stuff with headers and different versions of http
    response.from_info(info);
    //formats the response to protocol that the client will understand
    result.messages.emplace_back(response.to_string());
    return result;
}
```

We basically just have to define a `work` function/object/lambda whose signature matches what the API expects. The `worker_t::result_t` is the bit that `prime_server` will be shuttling around your architecture. The bulk of this function is just stuff you have to do at every stage in your pipeline. You'll need to unpack the message from the previous stage; in this case it was the server itself so the `work` function assumes its valid HTTP-looking bytes. You'll then want to formulate a response, either to be sent back to the client or forwarded to the next stage in the pipeline. In our case the workers respond to the client in all scenarios so we always initialize `worker_t::result_t::intermediate` as `false`. If it were `true` the worker would attempt to forward the result of this stage to the proxy for the next pool of workers. At the end we simply do some formatting to the response so that the client will make sense of it and we store this in `worker_t::result_t::messages`. OK so now what is left to do? Not much, just hook up some plumbing, basically constructing your pipeline.

```

int main(void) {
    zmq::context_t context;

    //http server, false turns off request/response logging
    std::thread server = std::thread(std::bind(&http_server_t::serve,
        http_server_t(context, server_endpoint, proxy_endpoint + "_upstream", result_endpoint),
        std::ref(context)),
        std::ref(context));

    //load balancer
    std::thread proxy(
        std::bind(&proxy_t::forward,
            proxy_t(context, proxy_endpoint + "_upstream", proxy_endpoint + "_downstream")),
        std::ref(context));
    proxy.detach();

    //workers
    auto worker_concurrency = std::max<size_t>(1, std::thread::hardware_concurrency());
    std::list<std::thread> workers;
    for(size_t i = 0; i < worker_concurrency; ++i) {
        //worker function could be defined inline here via lambda, it could be std::bind'd to
        //or simply just a free function like we have here
        workers.emplace_back(std::bind(&worker_t::work,
            worker_t(context, proxy_endpoint + "_downstream", "ipc:///tmp/NO_ENDPOINT", result_endpoint),
            std::ref(context)),
            std::ref(context));
        workers.back().detach();
    }

    //listen for SIGINT and terminate if we hear it
    std::signal(SIGINT, [](int s){ std::this_thread::sleep_for(std::chrono::seconds(1)); exit(0); });
    server.join();

    return 0;
}

```

First things first, all `zmq` communication requires a `context`. So we get one of those and pass it around to all the bits. Our setup is really simple, we run an `http_server_t` in one thread. The server keeps track of and forwards requests on to a load balancing `proxy_t` in another thread. The proxy keeps a queue of requests and shuttles them FIFO style to the first non-busy `worker_t` that it has in its inventory. We spawn a bunch of `worker_t`s which are constantly handshaking with the proxy. "I'm here" and "I'm done" messages let the proxy know which workers are bored and which are busy. This should minimize latency in so far as a greedy scheduler can. You'll notice the program is pretty much meant to be run as a daemon which is why it waits for `SIGINT`. `ctrl-c` it away when you are done with it.

Now we'll get your **R**esponsive **U**nicode **M**essages **P**ortal shaking.. err.. cracking.. err.. running.. with this:

```
g++ art.cpp -std=c++11 -lprime_server -o art
./art
```

From another terminal, you can hit it with curl to bask in your glorious, yet somewhat inappropriate, art work:

```
k@k:~$ for i in {0..7}; do curl "localhost:8002"; echo; done
(,_)_
(_|_)
(_*_)
(ㄣㄣ)
(ㄣiㄣ)
(ㄣεㄣ)
(,_)_
(_|_)
```

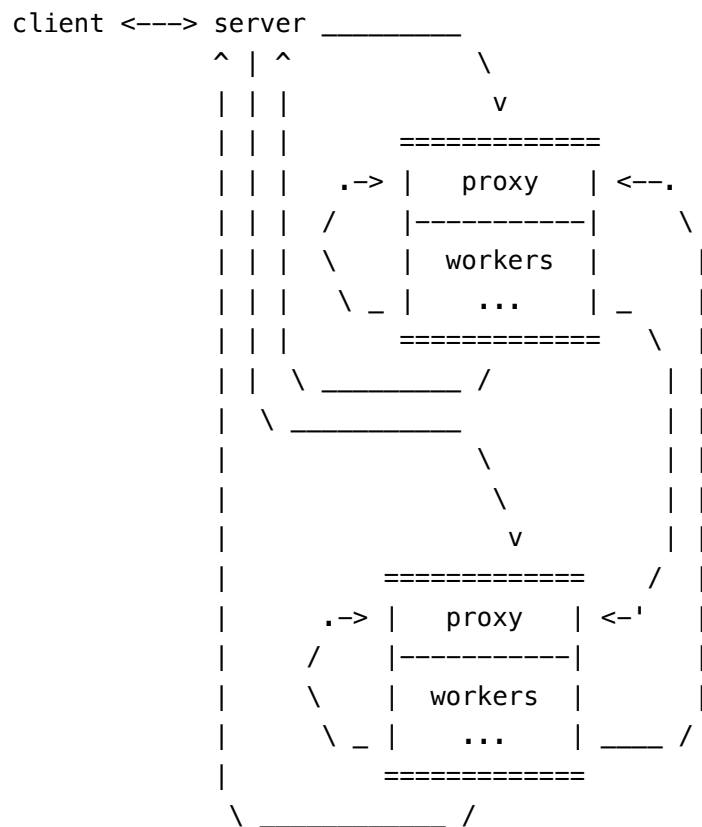
If you're interested in more sample code you can check out **Valhalla** (<https://github.com/valhalla>) or any of the sample daemon programs (src/*d.cpp) in the **prime_server** (https://github.com/kevinkreiser/prime_server) source.

The Future

The first thing we should do is make use of a proper HTTP parser. There are some impressive ones out there, notably **PicoHTTPParser** (<https://github.com/h2o/picohttpparser>) which is used in one of the webserver (**H2O** (<https://github.com/h2o/h2o>)) we came across in our searching. There may be a few issues with the streaming nature of the `ZMQ_STREAM` socket but they are worth working out so as not to have to maintain the mess of code required to properly parse HTTP.

The second thing we want to do is work `zbeacon` perks into the API. Currently the setup of a pipeline is somewhat cumbersome. Each stage must know about the previous and next (optional) stages as well as the loopback. It's clunky and requires a decent understanding to get it right. It's also very manual. With `zbeacon`, applications can broadcast their endpoints to peers so that they can connect to each other through discovery rather than via manual configuration. See **Hintjen's thoughts** (<http://hintjens.com/blog:32>) for a detailed description.

Automatic service discovery is pretty great, but that's not the really interesting part here; what if our pipeline weren't a pipeline? What if it were a graph?!



We may be reaching the limits of ASCII 'art' here but bear with me...

The implications are huge! We can remove the shackles of a the rigid fixed-order pipeline. Instead we can have application code determine the stages a request is forwarded to on the fly. For example, imagine you have a request that requires a bunch of iterations of a specific stage in the traditional pipeline. The only option you have is to perform the iterations on a single worker basically locking that worker until all the iterations are done. This would monopolize the worker; a certain request can ask for N iterations worth of work while the next request may only need one.

Allowing the stages to be connected in a graph structure (including cycles) would give the application the option to load balance portions of a larger request until the entire request has been fulfilled. Any problem that could be broken down into tasks of equal size (or at least more equal size) would have the potential to handle requests in a much more fairly balanced fashion.

A graph structure for the various stages also has the potential to better organize the functionalities of worker pools. One of the drawbacks of the fixed pipeline, mentioned earlier, was that to handle heterogeneous request types, you would either need workers that knew how to do more than one thing or indeed run the different types of requests on different clusters. That limitation would no longer exist in a graph structure. Based on the request type the application can just forward it to the relevant stage.

For example say you wanted to offer up math as a service (MaS of course). You might have:

- workers to compute derivatives
- workers to do summations
- workers to compute integrals

Now of course you could implement this all in a client side library, but for the sake of argument, ignore the impracticality for a second. What you wouldn't want to do is write a worker that does all three things. It would be nicer to isolate workers based on the type of work they perform (again the wishlist). This requires forwarding to a specific worker pool based on the url (in this example). Which brings up another `TODO`, we probably want to allow the server to forward requests to worker pools based on the URL (lots of other servers have this). Furthermore some of these operations are more complex than others. If you watched your system for a while (with a statistically relevant amount of traffic) you could look at the amount of CPU spent per stage and reallocate proportionally sized worker pools. You could even dynamically size the worker pools based on current traffic if you were really slick ;o)

The Conclusion

This has been a fantastic little experiment to have worked on. Even better it's been successful. We can claim that because it's used in at least **one production system** (<https://mapzen.com/projects/turn-by-turn>). Taking some excellent tools (ZMQ mostly) and building a new tool to help others build yet more tools is a very rewarding experience. If you think you may be interested in building a project/service/tool using this work, let us know! If you find something wrong submit an issue or better yet pull request a fix!

· 13 May 2016 ·



Kevin Kreiser

Kevin works on routing at Mapzen but secretly tries to work in all mapping disciplines. Er iss aa Pennsilfaanisch Deitscher.



Matt Amos

OpenStreetMap developer-contributor and chilli enthusiast. Writes vector tile and other data-mastication tools at Mapzen.

© 2017 Mapzen

It was the best of times. It was the worst of times.

search (/tag/search)

The Search team recently closed out another successful milestone. We integrated the Mapzen Who's on First gazetteer into our underlying Pelias engine. For more details on why and how we did that, check out **our previous post (who-s-on-first-it-s-mapzen-search/)**. As tradition calls in the software engineering realm, we held a post-mortem after-party! It's a long meeting dedicated to celebrating what we did right, and understanding and learning from everything else. For more details on the benefits and goals of post-mortem meetings, check out this **article (http://www.cdlib.org/cdlinfo/2010/11/17/the-project-post-mortem-a-valuable-tool-for-continuous-improvement/)**.

To prepare for this after-party, we set out to make a list of questions grouped by general themes to help drive discussion, then we wholeheartedly dove into each grouping. The notes and takeaways of post-mortems are typically kept private, if ever looked at again. Mapzen isn't a typical company, however, and we believe in the power of being *open* across *almost* everything we do, so why should our post-mortem findings be any different!

We're sharing our takeaways in order to generate discussion with our users and contributors. We want to be transparent so the community can understand our plans toward improvements and help us celebrate the things we already do really well. We would also love to get feedback from the community on what's working and what could be better. We're talking about process and communication here, not just geocoding features. To give you a preview of how post-mortem meetings typically go:

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way... – *Tale of Two Cities*, Charles Dickens

So let's dive right into our notes, shall we?! *We've included the questions along with corresponding groupings for context.*

General

Do we feel like we accomplished what we set out to accomplish from the beginning of the milestone? Did what we were building conceptually change at any point?

Overall we do feel like we accomplished our goals and didn't redefine the scope to any great extent. There were a few things we ended up pushing out of the milestone, but they didn't have a huge impact on the milestone goals. Tackling those things separately makes more sense anyway, in a more incremental fashion.

We ended up postponing the ability to return geometries from the `/place` endpoint and held off on making decisions about how to handle multiple administrative hierarchies for places.

It was hard to keep *on-task* because there were heaps of other stuff we wanted to work on, and in the end we had to choose what **not to do**, however, we don't feel like we took a lot off the table.

We also thought we would be able to sunset the use of Geonames in our service, but it proved to be a key dataset to many users, so we spent many additional cycles refactoring our Geonames importer and making sure that data was still available and better than ever. It was an unexpected twist and we dealt with it as best we could. In the future, we should determine the viability of sunsetting any feature or dataset more concretely before assuming it would be ok to remove it.

In addition to our actual goals, we ended up refactoring *MOST* of our code! There were parts that hadn't been touched in years and we did a lot of work to bring them all up to our current quality standards, along with proper tests... **hooray for testing!!!** This felt great and was necessary, but did have a significant impact on our timeline. In some cases, we weren't being honest with ourselves when estimating tasks, so it felt like some things dragged out beyond expectation. That made us sad. 😞

The awesome side-effect of the **Winter of Refactoring** is that it significantly raised team ownership and competency in ALL parts of our codebase. Just a few short months ago we welcomed two new members to the team, and as expected, learning the ropes in a large

distributed codebase takes time. With each team member rotating through the various components and all of us participating in frequent *deep-dive* sessions, we were able to get everyone up to the same level of code ownership in a much shorter period of time.

Planning

Reflect on size of iterations, effort estimation, scope and determine if we know enough now to make good decisions going forward?

Our biggest failure in this project was assuming the feature would be **straight-forward** and easy enough to do without a major upfront spec. We didn't break the large milestone into smaller bite-sized pieces and worked in a way that didn't allow for incremental releases. As things came up that would inevitably postpone the release, we were never in a release-ready state and had to keep putting off rolling any changes to production. The BIG release became bigger and scarier the more work went into it. **Let's never do that again!!!!**

Looking back, we could've released just the Who's on First importer first, bringing that new data into our build without changing all the other importers. Once that was out, we could've focused on the other existing importers, like OpenStreetMap, OpenAddresses, and Geonames, one at a time, then turn our attention to API updates to support the newly imported/updated data.

We have agreed as a team that going forward we will abide by the 2 week release cycle. If something can't be done in 2 weeks, it should be broken up in such a way that allows it to span over several 2 week release cycles. We hope the community of users and contributors will help keep us honest here! :)

During the integration process, we became extremely dependent on another team for some of the work. We were integrating with a brand new *moving-target* of a project. This added a level of complexity we didn't account for in our estimates. Even simple things, like cross-team communication, were often complicated because *timezones*, since our two teams span **Berlin-New York-San Francisco!**

Being the first consumer of a gazetteer certainly helped identify many areas of improvement on both sides of the equation. We consider that a great success and undervalued byproduct of this milestone. The team was also in agreement that the level and turnaround time of the support coming from the Who's on First team was really great and very much appreciated.

Process

Daily stand-ups? Too (in)frequent? Too long/short? Too (un)structured?

There was shared sentiment that we tend to get sidetracked during our daily standups. We've agreed to do our best to stay on track and stick to the traditional standup format of "what did you do yesterday?", "what will you do today?", "what are, if any, your current roadblocks?". We'll hold stand-up after-parties (you can tell we really like parties and like to keep them going as long as possible) to discuss things at a greater depth. Those not directly involved in after-party discussions can choose to stay or go, without judgement.

Did everyone feel connected / supported / informed enough to perform their assigned tasks?

Generally knowledge sharing and support from teammates has been at an all-time high! We've gotten into the regular habit of doing *deep-dive* sessions, where one person shares their screen and walks the others through a single feature they are working on, or we triage a reported issue in the same way. These weekly, if not more frequent, sessions have really elevated our ability to understand what everyone is working on beyond the superficial. It also allowed us to rotate through various parts of the code with more ease. We've probably at least doubled the **bus factor** (https://en.wikipedia.org/wiki/Bus_factor) numbers across all parts of the codebase. More *deep-dives*, FTW!

Stats and Insight

We've focused so much of our attention on improving search results and building out features, that we've neglected the supporting utilities and charting/logging. It's become evident that we need a proper dashboard that we can look at every stand-up. More insight into service performance, parameter usage, confidence levels of results, etc., would increase our confidence that we're making the right decisions. They would also allow us to track that those decisions are having the expected impact. We've already made this a priority for Q2 of 2016 and are working on making it a reality. We'll do our best to make sure those dashboards are public, so the community can benefit from the insight as well.

Development workflow

What are the highlights and pain points of our current build process, branching strategy, merging, and deploying to various environments?

During this milestone we were working with the following stacks, where a stack consists of an Elasticsearch cluster and an API server: - `development` stack, with a full world index - `staging` stack, which is used to build production quality Elasticsearch snapshots and test them before promoting to `production` - `production` stack

With only a single `development` server we weren't able to test features in a timely fashion. We've decided in order to ensure minimal delays during development and testing, we really need two `development` stacks. This will allow us to make changes to the Elasticsearch index during experiments, while still continuing to make API-only improvements. We agreed that we should only conduct **one** schema/index experiment at any given time.

We've also decided to use the `staging` stack to continuously run fresh builds. We currently only kick off a new build once a week unless something urgent comes up. We'd like to keep that stack busy with constant back-to-back builds that will only pause if something breaks or acceptance tests fail at the end of a build. Pausing after a problem gives us a chance to triage and restart the process manually once things are cleared up.

We talked a bit about our current branching strategy and decided that it warranted its own meeting. Stay tuned for some of the highlights from that in the future. This means data changes in OSM and other sources will be reflected in search much sooner: within 3 days instead of 7!

Get excited!!!

Outreach

Did we capitalize on opportunities to be public about our work? Did we do a good job supporting our growing community of users and contributors?

The general sense was that it's hard to blog about a work in progress. We did put out some great posts about **What Geocoding Means** (<https://mapzen.com/tag/geocoding/>) and how we envision it done right. We'll be shooting for a much more predictable blogging cycle. Ideally, we'll have something interesting, even if small, to share with the community. Team members will take turns being on the hook for a post.

One major concern that surfaced around outreach is that external contributors need more love!!!! We've been lucky to have some great pull requests come through during this milestone and we dropped the ball on responding to those within a reasonable timeframe. We decided it's really important to respond quickly to a PR, especially if we don't think it's something we can ever merge or it doesn't fit in with our current roadmap.

We realized that it's important to clearly outline our expectations for contributions: be clear about the importance of providing unit tests and including a solid description of what feature the changes are implementing/fixing.

I don't know about you, but it felt great for the team to get closure and have some plans for what's next. We all went out to celebrate with a round of mini-golf on Pier 25 in NYC!



Cover image sourced from **The Victorian Web**

(<http://www.victorianweb.org/art/illustration/barnard/ttc/20.html>). Scanned image and text by **Philip V. Allingham** (<http://www.victorianweb.org/misc/pvabio.html>)

· 18 May 2016 ·

Diana Shkolnikov



Diana is the engineering director of Search@Mapzen and a proponent of mandatory fun, testing, education, and paleo, not necessarily in that order.

© 2017 Mapzen

Targeted Editing – Cycleways

targeted-editing (/tag/targeted-editing) **osm** (/tag/osm)

Maps are current as of Oct 2016. To see new data, check out Mapzen's **bike map** (<https://mapzen.com/blog/bike-map/>) style.

Welcome to the Targeted Editing series where today you can kick back on your favorite sectional, or make OpenStreetMap data exceptional! (Both take about the same amount of time.)

May is **National Bike Month** (<http://www.bikeleague.org/bikemonth>) in the United States, and bike to work and school activities are being hosted across the country. Are you a cycling commuter or aspire to be one? You are in **good company** (<http://www.census.gov/content/census/en/library/visualizations/2016/comm/bike2work/jcr:content/map.detailitem.800.medium.jpg/1461767042769.jpg>)!

Read on for some basic uses for tags commonly associated with bike maps, and how you can check to see if you can use your **local knowledge** to add features where they are needed.

How are bike paths and associated features used?

1. **Help with search.** What kinds of things might bicyclists search for? Cyclists definitely search for routes that are safe but there are a lot of additional features that are equally important. Just a few examples include bicycle shops for supplies, bicycle repair stations, and definitely bicycle parking.
2. **Help with routing.** Getting from point A to B via bicycle can incorporate road criteria that pedestrians and cars prioritize differently. The surface type of the road can be used to differentiate routes for different types of bicycles. Road types or road classes can be used to differentiate routes based on the experience level of the rider. Even the type of bicycle lane can make a huge difference in the desirability of a suggested route. Some bicycle lanes are protected, some are merely buffered from vehicle traffic, and others are completely separated, bypassing vehicle traffic all together.
3. **Help with map rendering.** Bicycle features can be used to make custom maps designated specifically for bicycle commuters.

Are there bicycling features in OpenStreetMap? Yes!

A quick review of the data in some of our favorite bike friendly cities shows how promising it is in more detail:

Bicycle Features in OpenStreetMap

	Bike Tracks (km)	Bike Lanes (km)	Bike Route Relations (km)	Bike Points of Interest
Amsterdam	6,993	1,061	5,046	562
Barcelona	455	326	110	276
Berlin	4,338	3,207	14,320	1,032
Buenos Aires	148	204	294	115
Chicago	3,201	833	2,187	438
Copenhagen	5,381	2,983	3,411	684
London	3,831	2,149	9,147	1,166
Minneapolis / St Paul	1,733	731	2,159	218
Montreal	2,032	2,152	1,757	175
New York City	654	814	1,613	492
Paris	2,189	2,385	1,465	1,673
Portland OR	202	2,621	1,199	146
San Francisco	553	1,238	1,262	175
Tucson	241	156	505	25
Washington DC	2,519	1,100	1,722	337

*SQL queries (https://github.com/mapzen-data/targeted-editing/blob/gh-pages/queries/bicycle_sql.sql) for those interested in generating stats for additional cities.

For our table above, bike tracks are *separate from the road* or simply off-road, and bike lanes are *part of the existing road*. Some of them are part of bike route relations, but we would like to call those out separately. Bike points of interest includes bicycle parking, repair stations, rental locations and shops that specialize in bicycle merchandise.

Bicycle infrastructure is well represented in OpenStreetMap and there are lots of web maps that have been built to highlight these features. Here are just a few:

- **Bike Citizens** (<https://map.bikecitizens.net>)
- **CycleStreets** (<http://www.cyclestreets.net/getmapping/>)
- **cycle.travel** (<http://cycle.travel>)
- **Global Bike Map** (http://doodles.mountainmath.ca/bike_global.html) built by **Jens von Bergmann** (<http://doodles.mountainmath.ca/blog/2016/05/16/my-global-bike-map/>)
- **Hike & Bike Map** (<http://hikebikemap.org>)
- **OpenCycleMap** (<http://www.opencyclemap.org>)
- **OpenFietsmap** (<http://www.openfietsmap.nl>)
- **OpenRouteService** (<http://openrouteservice.org>)
- **VeloMap** (<https://www.velomap.org>)

Let's Get Mapping!

If you are a bicycle enthusiast you will find lots of tags in OpenStreetMap to support cycle life. Most of the popular tags are associated with the highway and cycleway keys, but other features associated with bike rentals, parking, and repairs are growing in numbers. Read on for some suggestions and links to their corresponding OpenStreetMap wiki pages.

Some OpenStreetMap conventions to keep in mind:

1. Do not digitize separate features for bike lanes if they are part of a road with no separation from vehicular traffic.
2. Designated **Cycle Lanes**
(http://wiki.openstreetmap.org/wiki/Key:cycleway#Cycle_lanes) tagged `cycleway=lane` are on the existing road.
3. **Shared Lanes**
(http://wiki.openstreetmap.org/wiki/Key:cycleway#Shared_cycle_lanes) tagged `cycleway=shared_lane` , `cycleway=shared_busway` , or simply `cycleway=shared` indicate shared cycle lanes.
4. **Cycle Tracks** (http://wiki.openstreetmap.org/wiki/Key:cycleway#Cycle_tracks) running parallel to the road yet separated from cars by a physical barrier are tagged `highway=cycleway` *when they are digitized as separate features*.

Creating a great map for cyclists involves identifying preferred routes. When given the choice between riding a bike on a road with a designated bike lane vs a road without, most cyclists will choose the former. We're not talking about the mighty few that live for the adrenaline of riding in full traffic. We know you are happy to ride just about anywhere. For the rest of us, a little protection goes a long way towards a comfortable safe ride.

In addition to features that support routing, there are also several points of interest that are particularly valuable to cyclists. Parking designated for bicycles and shops that specialize in bicycle related merchandise are just a few of the things that cyclists would love to see prominently featured on a map.

Cycle Based Characteristics

Have you ever had the pleasure of riding a bike on a designated bicycle route? There are some excellent routes that incorporate reclaimed rail corridors. The **Midtown Greenway** (https://en.wikipedia.org/wiki/Midtown_Greenway) in Minneapolis is a great example. Since these routes were originally traversed by trains, they tend to be separate from vehicular traffic and gentle in grade. Perfect for pleasurable cycling!

If you are aware of a designated and often signposted bicycle route, check OpenStreetMap to see if it's there. It will likely and should be represented as a relation.

route=bicycle (<http://wiki.openstreetmap.org/wiki/Tag:route%3Dbicycle>)

There are several tags that enhance bike route relations. For example, make sure the bike route relation has a name so it can be search for, displayed, and incorporated into spoken and written turn by turn directions.

name=* (<http://wiki.openstreetmap.org/wiki/Key:name>)

There are also different types of bike routes relative to the scope of the route. Is it a national, regional, or local bike route? Explore these cycle network tag options for **n**ational **c**ycle **n**etworks, **r**egional **c**ycle **n**etworks, and **l**ocal **c**ycle **n**etworks:

network=ncn (<http://wiki.openstreetmap.org/wiki/Tag:network%3Dncn>)

network=rcn (<http://wiki.openstreetmap.org/wiki/Tag:network%3Drcn>)

network=lcen (<http://wiki.openstreetmap.org/wiki/Tag:network%3Dlcen>)

Really, the more detail you can add, the better! All of these details can be incorporated into generating the best routes and beautifully detailed maps.

If you prefer routes that are designated solely for bicycles, tags make all the difference. Do you ride on a multi-use path that segregates bikes from pedestrians? Find it in OpenStreetMap and ensure that the `bicycle=designated` with `segregated=yes` tags are associated with the associated features tagged `highway=path`.

For multi-use paths, there are many additional **combinations**

(http://wiki.openstreetmap.org/wiki/Tag:highway%3Dpath#Usage_as_a_Universal_Tag) of tags to explore.






Example	Mapping	Description
	<code>highway=path</code> + <code>foot=designated</code> + <code>bicycle=designated</code> + <code>segregated=no</code>	Signposted foot and bicycle path
	<code>highway=path</code> + <code>foot=designated</code> + <code>bicycle=designated</code> + <code>segregated=yes</code>	Signposted foot and bicycle path with dividing line.
	<code>highway=path</code> + <code>foot=designated</code> + <code>bicycle=designated</code> + <code>mofa=yes</code>	Signposted foot and bicycle path with additional permission of mopeds.
	<code>highway=path</code>	A generic multi-use path open to non-motorized vehicles, implicitly allowed for pedestrians, cyclists, ..., see OSM_tags_for_routing/Access-Restrictions

image via **OSM wiki**

(http://wiki.openstreetmap.org/wiki/Tag:highway%3Dpath#Usage_as_a_Universal_Tag)

Sure, there are often conventions for which side of the road to ride on relative to vehicle traffic, but sometimes a bike lane itself can be strictly designated oneway for bikes. Yes, it's true! But how do you tag this so it's relative to the bike lane, NOT vehicular traffic? This is where a subkey comes into play. A subkey is represented with a colon. It takes the form `key:subkey=value`, and in this case, the values are either yes or no.

`oneway:bicycle=*` (http://wiki.openstreetmap.org/wiki/Bicycle#Bicycle_Restrictions)

Once this subkey is in place, the flow of traffic is relative to the direction in which the *way* was digitized. Luckily, the direction of the way can be selected and reversed. Just select the feature and look for this icon in ID: 

Subkeys are also used to indicate which side of the road a cyclist can find a bike lane. These tags are particularly helpful when a lane is only present on one side. An example would be `cycleway:right=lane` paired with a `highway` tag.

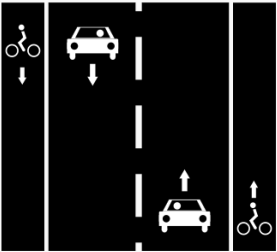


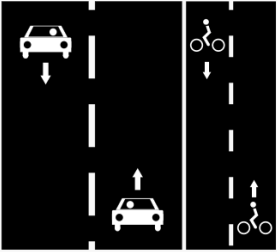


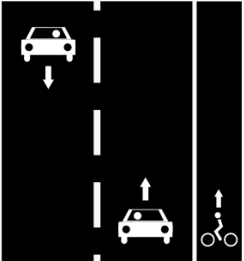

Ref	Context	Photo	OSM	Description
L1a				Cycle lanes on left and right sides of the road. Way A : <code>highway=*^[1] + cycleway=lane</code> (recommended) or Way A : <code>highway=*^[1] + cycleway:left=lane + cycleway:right=lane</code> or Way A : <code>highway=*^[1] + cycleway:both=lane</code>
L1b				Bidirectional cycle lane on right side of the road. Way A : <code>highway=*^[1] + cycleway:right=lane + oneway:bicycle=no</code> (recommended) (nb: the <code>oneway:bicycle=no</code> tag should be interpreted to refer to the cycle lane in this case, as the highway itself is per default bidirectional for all transport modes anyway) or Way A : <code>highway=*^[1] + cycleway=lane</code> (not recommended, as this can't be distinguished from L1a)
L2				Oneway cycle lane on right side of the road only. Way A : <code>highway=*^[1] + cycleway:right=lane</code> (nb: bikes can use the normal highway on the left side)

image via **OSM wiki**
(http://wiki.openstreetmap.org/wiki/Bicycle#Cycle_lanes_in_bidirectional_motor_car_road_s)

Who else is rolling down the bike path with you? Are electric bikes allowed? Be sure to check local laws. California just passed a **new law** (<http://www.peopleforbikes.org/blog/entry/new-e-bike-law-passes-in-california>) outlining when it is permissible to operate an electric bicycle in

bicycle lanes.

moped=* (<http://wiki.openstreetmap.org/wiki/Key:moped>) mofa=* (<http://wiki.openstreetmap.org/wiki/Key:mofa>)

Points of Interest for Cyclists

You are ready to ride, but you don't even have a bike! Lots of lucky city dwellers have bicycle share systems. San Francisco has the **Bay Area Bike Share** (<https://www.sfbike.org/our-work/regional-advocacy/bike-share/>).



image via **the author** (<https://github.com/IndyHurt>)

Chicago area residents have **Divvy** (<https://www.divvybikes.com>), Paris has **Vélib'** (<http://www.velib.paris>) and **Santander Cycles** (<https://tfl.gov.uk/modes/cycling/santander-cycles>) is well known in London.

What bike share operators do you have in your city, and can they be added to OpenStreetMap? Of course! Just remember that OpenStreetMap is comprised of local knowledge and truly open data, so feel free to add the locations you are personally familiar with. Express permission to use the bike station maps provided by these operators must be obtained before using them to add features to OpenStreetMap. When this permission is granted, be sure to include a **source=*** (<http://wiki.openstreetmap.org/wiki/Key:source>) key with your edits.

amenity=bicycle_rental

(http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dbicycle_rental)

operator=* (<http://wiki.openstreetmap.org/wiki/Key:operator>)

The table above aggregates several types of bicycle points of interest, but if we focus on just bicycle share and rental systems, you'll see how often they make up the bulk of the points. Paris sure does have an impressive number of bike share locations in OpenStreetMap - nearly 1,300. For context, the Vélib' bike share company in Paris boasts 1,800 stations on their website (as of May, 2016). That puts OpenStreetMap at roughly 70% coverage. Truly impressive!

Now you are riding along and suddenly you get a flat tire, or your brakes need adjusting, or maybe it's the derailleur this time. Perhaps you have all the tools with you, but if not, you'll be happy to find a bike shop or a bike repair station. I just recently saw a bike repair station next to

a multi use path, but I can't remember where for the life of me. If only I had added it to OpenStreetMap right when I saw it...

shop=bicycle (<http://wiki.openstreetmap.org/wiki/Tag:shop%3Dbicycle>)

amenity=bicycle_repair_station
(http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dbicycle_repair_station)

You've reached your destination and what is the first thing you look for? Did you say "a safe place to park and lock up your bike"? Sometimes you can just lock your bike up to any immovable structure. This is definitely **NOT** the case on most school campuses! Let's add designated parking for bikes:

amenity=bicycle_parking
(http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dbicycle_parking)

capacity=* (<http://wiki.openstreetmap.org/wiki/Key:capacity>)

Road Based Characteristics

Sometimes the rules of the road also influence the comfort of the ride for cyclists. Pedalling alongside fast moving traffic can be terrifying if you are not an experienced rider with nerves of steel. Routes can be tailored for cyclists based on experience. A route generated for a less experienced bicycle rider might avoid busier secondary roads and favor residential roads, or simply favor roads that have lower speed limits for vehicle traffic.

Bicycle routes can also be customized based on the type of bike you have. If you have a road bike, routes can be restricted to roads that are paved. If you have a mountain bike, this restriction may not apply. The `surface` tag stores this information.

highway=* (<http://wiki.openstreetmap.org/wiki/Key:highway>) **primary**
(<http://wiki.openstreetmap.org/wiki/Tag:highway%3Dprimary>) **secondary**
(<http://wiki.openstreetmap.org/wiki/Tag:highway%3Dsecondary>) **tertiary**
(<http://wiki.openstreetmap.org/wiki/Tag:highway%3Dtertiary>) **residential**
(<http://wiki.openstreetmap.org/wiki/Tag:highway%3Dresidential>)

maxspeed=* (<http://wiki.openstreetmap.org/wiki/Key:maxspeed>)

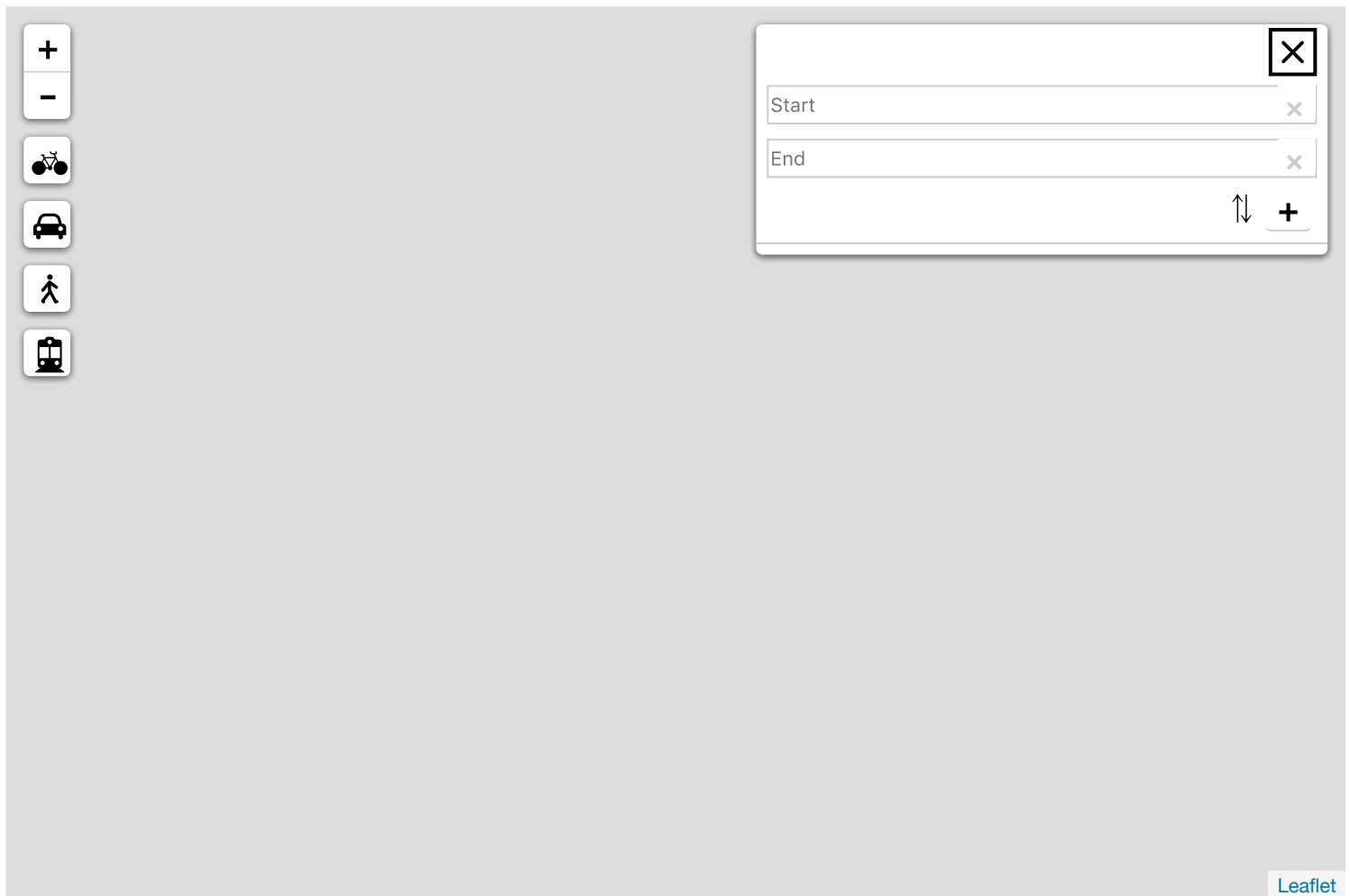
surface=* (<http://wiki.openstreetmap.org/wiki/Key:surface>)

Where to Start

The best place to start is always where you have local knowledge. If you are familiar with a bicycle infrastructure that isn't tagged properly in OpenStreetMap, now is your chance to remedy the situation!

Interactive Map

Would it be helpful to see where bicycling infrastructure exists in OpenStreetMap? There's a map for that! This simple map allows you to enter in a start and end point and generate a bicycle route. Further refine the route by dragging the start and end icons.



This map is interactive! Open full screen ↗ (<https://mapzen-data.github.io/targeted-editing/te-cycleways/>)

Check out bike infrastructure in **Paris** (<https://mapzen-data.github.io/targeted-editing/te-cycleways/#point0lat=48.8537&point0lng=2.3493&point1lat=48.8599&point1lng=2.3250&mode=bicycle>), **Copenhagen** (<https://mapzen-data.github.io/targeted-editing/te-cycleways/#point0lat=55.6567&point0lng=12.5687&point1lat=55.6785&point1lng=12.5927&>

mode=bicycle), London (<https://mapzen-data.github.io/targeted-editing/te-cycleways/#point0lat=51.5145&point0lng=-0.1222&point1lat=51.5070&point1lng=-0.1059&mode=bicycle>), Shanghai (<https://mapzen-data.github.io/targeted-editing/te-cycleways/#point0lat=31.2445&point0lng=121.4849&point1lat=31.2214&point1lng=121.4814&mode=bicycle>) and more!

Custom symbology derived from the underlying OpenStreetMap tags has been employed to help identify how features are currently represented in the data. We are hoping that the labels, line, and color choices will help you recognize bike infrastructure and identify whether or not it is tagged properly. Special thanks to **Nathaniel V. Kelso (<https://github.com/nvkelso>)** for building this special map style for this blog post!

The map above uses Mapzen **Search (<https://mapzen.com/projects/search/?lng=-122.01170&lat=37.91370&zoom=12>)**, **Turn-by-Turn Routing (<https://mapzen.com/projects/turn-by-turn/>)**, **TransitLand (<https://transit.land>)** for public transit, and **Tangram (<https://mapzen.com/projects/tangram/>)** to display routes generated for drivers, pedestrians, public transportation users and cyclists. You can build amazing things with these services, too! Head over to our **developer (<https://mapzen.com/developers/>)** page to sign up for API keys.

What can you do if you plug in a route with any of the solutions above and get a route that doesn't quite fit with your local knowledge of the area?

Your best bet is to jump right in to OpenStreetMap and investigate the data to see where a tag might be missing or where a connectivity issue may exist.

To edit features, hover over bike infrastructure or bike points of interest in the map for this post to bring up an info bubble with links to editing tools that can be used to modify features.

Is something missing all together? Shift-click the map to launch the iD editor for OpenStreetMap at that location. You may not have control over how the software generates the route, but you always have the ability to investigate the data and pass on feedback to developers when the data seems sound.

Getting Started with OpenStreetMap

Need instructions on how to edit with iD? Here are some links to outstanding tutorials from LearnOSM, the OpenStreetMap wiki, and the United States Department of State's Humanitarian Information Unit:

- **LearnOSM** (<http://learnosm.org/en/>)
- **OSM Beginners' Guide** (http://wiki.openstreetmap.org/wiki/Beginners'_guide)
- **MapGive Learn to Map** (<http://mapgive.state.gov/learn-to-map/>)
- **Missing Maps Video Tutorials** (<http://www.missingmaps.org/contribute/#learn>)

Are you a mapping wiz and interested in a more advanced editor? Try out **JOSM** (<https://josm.openstreetmap.de>) with **excellent documentation** (<https://www.mapbox.com/blog/osm-mapping-guide/>) from Mapbox.

Editing with a Mobile Device

For those bike stores and other points of interest, I bet you are interested in a mobile app to edit OpenStreetMap while you are on the go.

iOS users can check out **Go Map!!** (<https://appsto.re/us/dafwj.i>) by Bryce Cogswell. This free editor allows you to add features and edit existing features. Check out the **Go Map!!** (http://wiki.openstreetmap.org/wiki/Go_Map!!) wiki page for more details. **Pushpin** (<http://www.pushpinosm.org>) by Fulcrum is another excellent iOS editor for OpenStreetMap points of interest.

Looking for an Android equivalent? **Vespucci** (<https://play.google.com/store/apps/details?id=de.blau.android>) by Marcus Wolschon is a great option. Check out the **Vespucci** (<http://vespucci.io>) home page for documentation and tutorials.

Last but not least, for both iOS and Android, **MAPS.ME** (<http://maps.me/en/home>) now **supports editing** (<http://blog.maps.me/2016/04/lets-make-most-detailed-map-together.html>).

Get outside, increase your local knowledge, and share it with the world through your OpenStreetMap edits!

Check out all the posts in the **Targeted Editing series** (<https://mapzen.com/tag/targeted-editing>):

- **Airports** (<https://mapzen.com/blog/targeted-editing-airport-polygons>)
- **Banks** (<https://mapzen.com/blog/new-years-resolutions-money/>)
- **Building Names** (<https://mapzen.com/blog/targeted-editing-name-that-building>)
- **Campus Mapping** (<https://mapzen.com/blog/targeted-editing-campus-mapping/>)
- **Cycleways** (<https://mapzen.com/blog/targeted-editing-cycleways/>)
- **Fitness** (<https://mapzen.com/blog/new-years-resolutions-fitness>)
- **Groceries** (<https://mapzen.com/blog/new-years-resolutions-groceries>)

- **Hospitals** (<https://mapzen.com/blog/targeted-editing-hospital-polygons>)
- **Schools** (<https://mapzen.com/blog/targeted-editing-school-polygons>)
- **Stadiums & Parking** (<https://mapzen.com/blog/targeted-editing-tailgate-mania>)
- **Street Names** (<https://mapzen.com/blog/targeted-editing-no-name-roads>)
- **Transit Colours** (<https://mapzen.com/blog/targeted-editing-transit-colours>)
- **Travel & Lodging** (<https://mapzen.com/blog/new-years-resolutions-travel>)

· 19 May 2016 ·



Indy Hurt

Indy was our resident data scientist lending her geographic expertise to all things "open".

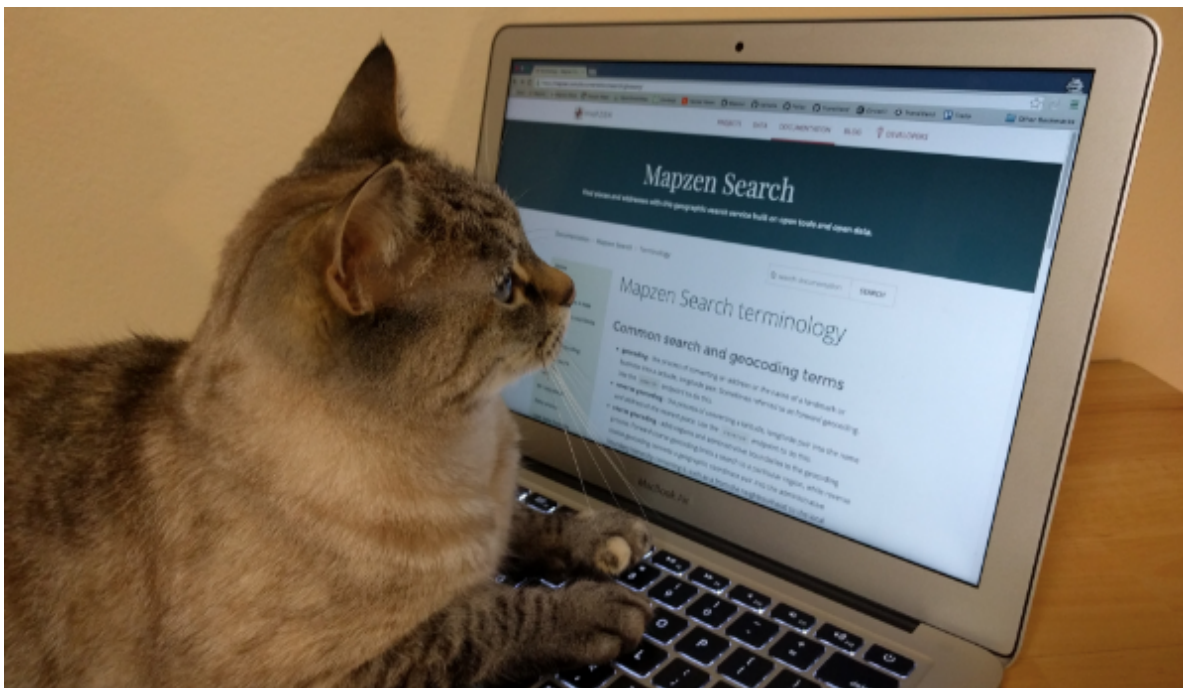
© 2017 Mapzen

Contribute to open-source projects through documentation

documentation (/tag/documentation)

Writing documentation is a good way to start contributing to an open-source project. Documentation may have been overlooked or written as an afterthought during the software development process, so there are usually many opportunities for you to help enhance the content.

Users of all experience levels and skills have perspectives that are valuable additions to the documentation. By contributing your knowledge, you are making the software more useful and accessible to everyone.



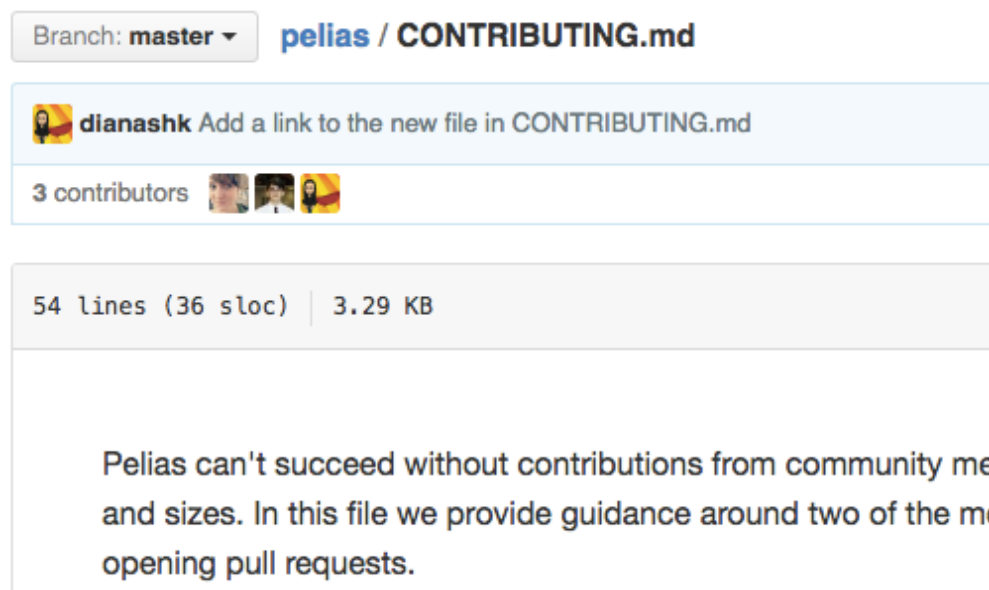
Bailey the cat thinks the help needs more information about yarn and boxes.

Where to start

The code and accompanying documentation for many open-source projects are stored on **github.com** (<https://github.com>), which is a website that enables project collaboration. If you are new to GitHub, consider making your first open-source contributions to documentation or

the README file because they likely have simpler setup requirements than attempting to modify code. Even if the code is mature or closely guarded, its documentation may be more welcoming to modifications. You need to create a **GitHub account** (<https://help.github.com/articles/signing-up-for-a-new-github-account/>) to get started.

Sometimes, you will find documentation files alongside the code, while other projects have a separate repository (the term for the grouping of files, folders, and data that makes up a project) for documentation. When you find the repository you want to work on, review the **contributor guidelines** (<https://github.com/blog/1184-contributing-guidelines>) to learn about maintaining a positive community environment and the process to follow to propose changes.



Contributor guidelines for the Pelias project, which is an open-source geocoder that powers Mapzen Search.

When you are ready to propose updating the source file with your changes, open a **pull request** (<https://help.github.com/articles/using-pull-requests/>). A pull request is the way you notify others about your suggestions. The community managers then review your changes and decide whether to accept them into the source. This feedback helps them discover what is important to users based on what the community is looking at and are updating in the documentation.

What to write

While reading the documentation, you might find simple errors like typos and broken links that could be resolved relatively quickly. Other times, you might encounter a tutorial step, API version number, or screenshot that is out of date. Unfortunately, you might hit something that is

completely wrong or missing. You can fix these yourself if you have time, or report problems or enhancement requests to the project's managers through **GitHub issues** (<https://help.github.com/articles/creating-an-issue/>).

Beyond fixing issues in existing documentation, adding new content is another excellent contribution. If you are an advanced user, perhaps you can write tips, shortcuts, and detailed workflows. As a beginning user, share what you learned to make the ramp-up process easier on others. Adding tutorials, enhancing installation instructions, giving sample code, and building troubleshooting sections are very helpful. In addition, translating documentation into another language can bring a project to a new set of users.

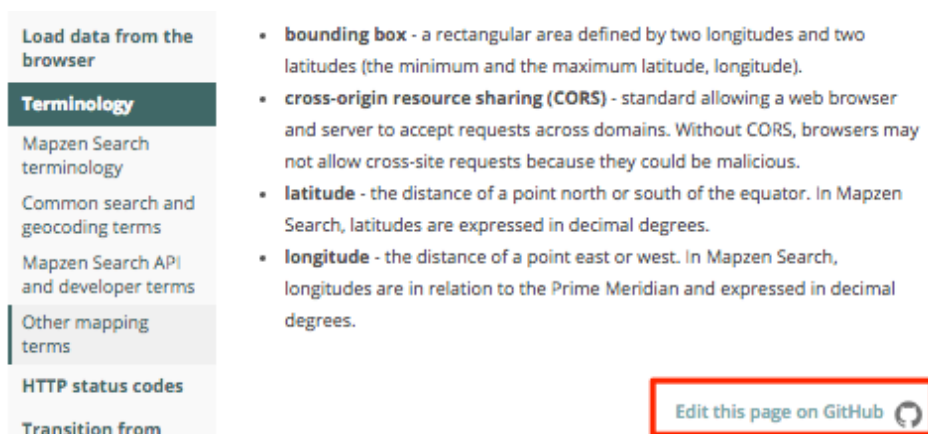
You might also look for GitHub issues that have been tagged with Help Wanted, which indicates external assistance is requested. These lists can provide suggestions on what to do when you want to help an open-source project but are not sure where to start.

How to update Mapzen's open documentation

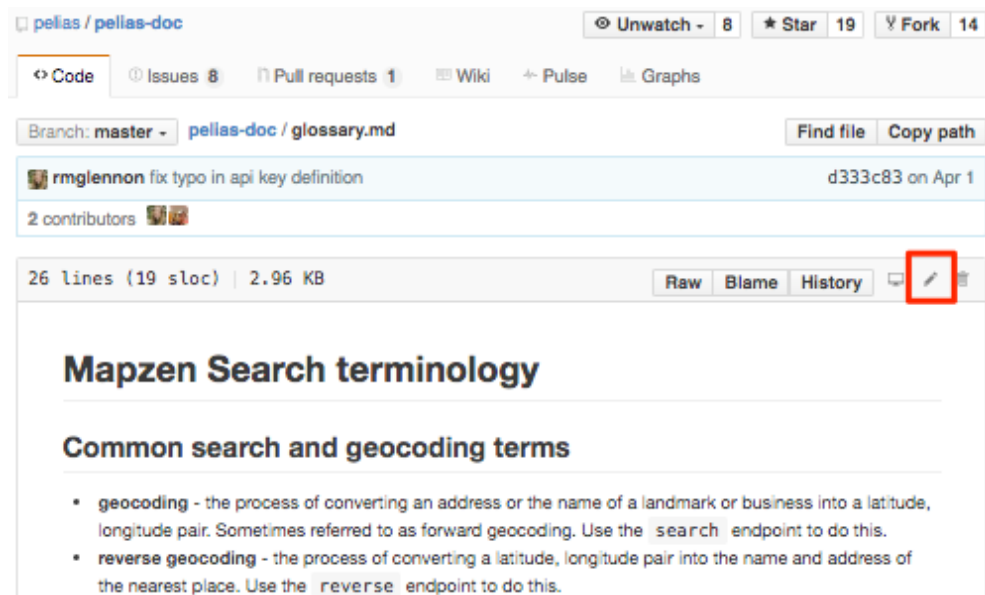
You can find Mapzen's technical documentation at <https://mapzen.com/documentation> (<https://mapzen.com/documentation>). The source files used to build the site are stored as **markdown (.md) files** (<https://en.wikipedia.org/wiki/Markdown>) on GitHub, and **are formatted** (<https://mapzen.com/blog/doc-site/>) into what you see on Mapzen's website.

The bottom of every page on the Mapzen documentation site has a link where you can view and edit the source file. Follow these steps to contribute to the help.

1. On any help topic, click Edit this page on GitHub to access the source markdown file.



2. Sign in with your GitHub account.
3. Click Edit this file to enter a mode where you can modify the page. GitHub automatically creates a copy that you can edit.



pelias / pelias-doc

Unwatch 8 Star 19 Fork 14

Code Issues 8 Pull requests 1 Wiki Pulse Graphs

Branch: master pelias-doc / glossary.md Find file Copy path

rmglennon fix typo in api key definition d333c83 on Apr 1

2 contributors

26 lines (19 sloc) | 2.96 KB Raw Blame History

Mapzen Search terminology

Common search and geocoding terms

- geocoding** - the process of converting an address or the name of a landmark or business into a latitude, longitude pair. Sometimes referred to as forward geocoding. Use the `search` endpoint to do this.
- reverse geocoding** - the process of converting a latitude, longitude pair into the name and address of the nearest place. Use the `reverse` endpoint to do this.

4. Add your updates and save them (also known as making a commit into your fork of the file).

5. Open a pull request when you are ready to propose your changes.

There have been dozens of community contributions to Mapzen's documentation so far! Looking through the documentation repositories, you have submitted pull requests resolving typos, formatting problems, and unclear API input parameter descriptions.

In addition, you have logged GitHub issues relating to API enhancements, unexpected API responses, and differences between the text of the documentation and the actual API result. You have also posted questions about how to localize the API text string values because you want to help with translating them. Some of the documentation issues you reported were traced to problems in the software, so both the documentation and code were improved with your help.

Thanks to all of you who have contributed to Mapzen's documentation so far—and the documentation for other projects. Community support is fundamental to an open-source project's success.

· 27 May 2016 ·



Rhonda Glennon

Rhonda is Mapzen's technical publications manager and writes about maps and developer tools.

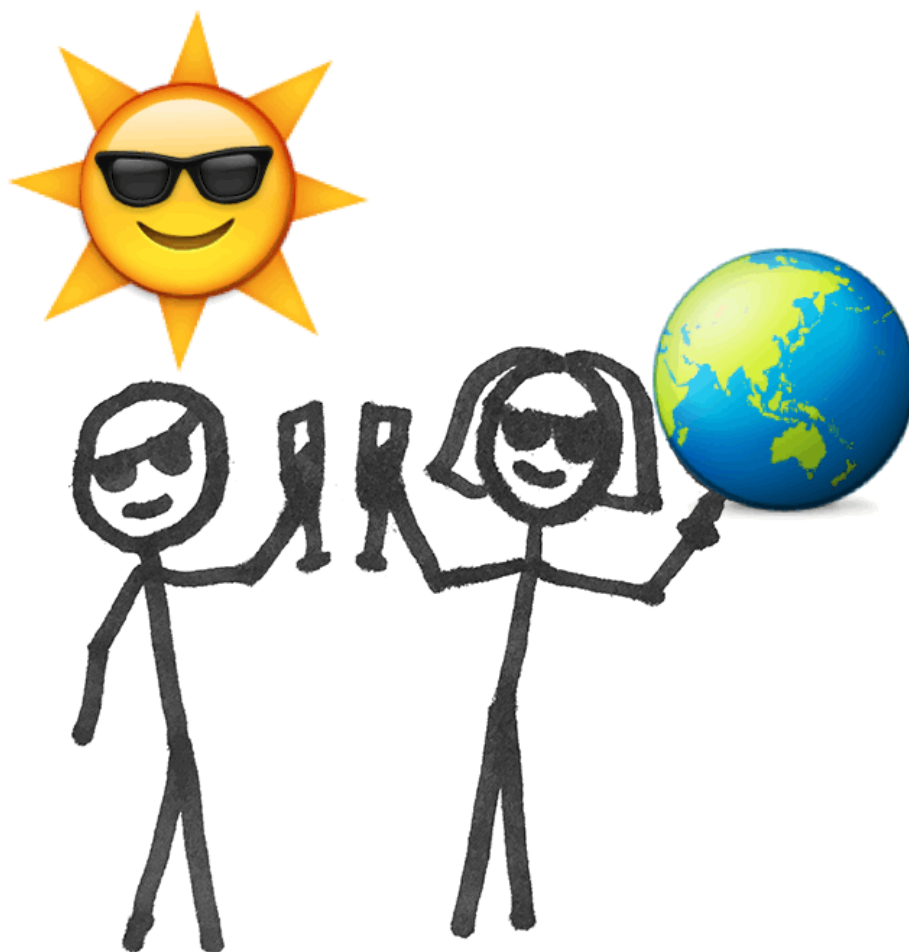
© 2017 Mapzen

Summer Office Hours – Working with Metro Extracts!

osm (/tag/osm)

Welcome to Summer! To celebrate we at Mapzen invite you to don your white pants and join us for our first Summer Office Hours!!! For this very first one, we will BBQ* some open data and get to the meat of **Metro Extracts** (<https://mapzen.com/data/metro-extracts/>)!

(**OpenStreetMap** (<http://www.openstreetmap.org/>) data served up in easy byte-sized pieces)
#thankOSMgods



yay summertime!

Come with your Metro Extracts questions and we'll have experts of all technical abilities to get you started on making your own map!

The deets:

- When: Friday, June 3rd
- Time: 9:00am - 11:00am
- Place: Mapzen (Samsung Accelerator) Offices | 30 W 26th St, 7th Floor, New York NY 10010
- **RSVP here (<http://goo.gl/forms/ufvalrlod5y9g5QC2>)**

Of course, if you need help at any time, or want us to hold office hours in your city, send us a note at **hello@mapzen.com** (**<mailto:hello@mapzen.com>**)!

[*There will be summer breakfast, not summer BBQ]

· 31 May 2016 ·



Ekta Daryanani

Lead, Design + User Experience. Thinks: colors, people, community and city life. Does: mobile, maps, rock climbing and yoga.

© 2017 Mapzen

Take the Bus, Gus. No looking back, Jack.

[transitland](#) ([/tag/transitland](#)) [routing](#) ([/tag/routing](#)) [mobility](#) ([/tag/mobility](#))

From the coast to the mountains, from the river to the sea, from the plains to the train, Mapzen is opening transit routing to the world.

Over the past few months, public-transit agency staffers and freelance data enthusiasts have submitted hundreds of GTFS feeds to **Transitland** (<https://transit.land>), our collaborative effort to aggregate and improve transit data. As an open data platform, Transitland is intended to power apps, visualizations, and analyses by anyone and everyone. As part of **the Mapzen Mobility initiative** ([/blog/introducing-mapzen-mobility/](#)), we've also been busy building open-source apps and services on top of Transitland ourselves.

Today we're excited to share that **Mapzen Turn-by-Turn** ([/projects/turn-by-turn](#)) can plan multimodal journeys in over 200 regions around the world, using Transitland data. This makes Mapzen Turn-by-Turn the first open-source routing engine that can plan multimodal journeys (almost) anywhere in the world—and using only open and freely available data.

Easy Options for Developers

With this international coverage, we hope to help even more developers focus on making their own apps and digital experiences—not collecting, licensing, munging, and wrestling data—and now they have useful options available to them for working with open transit data:

- Use the **Transitland Datastore API** (<https://transit.land/documentation/datastore/>) to access information about a transit operator, a stop location, or schedules at a certain time of day.
- Use Mapzen Turn-by-Turn to plan journeys in **the 200+ feeds on Transitland that are marked as “routable.”** (https://transit.land/feed-registry?import_level=4)

Even Easier Demos for Everyone

Getting a JSON response from an API is a dependable way to power an app or a visualization, but yes, we know it isn't that visceral of a way to experience, well, anything. Here's an interactive drag-and-drop demo that shows off the dense transit coverage in San Francisco, Prague, New York City, Rome, Los Angeles and Vancouver. Try moving those Point A and Point B pins and see what routes, connections and schedules are possible, then advance through the slides to try another city.

Mapzen Turn-by-Turn

Demo

+

-

Leaflet (<http://leafletjs.com>)

NYC

Rome

Philadelphia

Sea to Sky

Want to hack this demo for your own purposes? It's only a two-step process:

- Open up **your web browser's developer tools** (<http://debugbrowser.com/>) so you can see the demo code making requests to the Mapzen Turn-by-Turn API.
- Sign up for **a free Mapzen developer key (/developers)** so you can submit requests like that from your own website, mobile app, or visualization/analysis script.

Actually, there's a third, optional step:

- If you look closely, you'll see that demo also looks up Point A and Point B using **Mapzen Search (/projects/search)**. It's that reverse-geocoding process that allows us to display a street address and a city name, rather than raw latitude and longitude. Mapzen Turn-by-Turn plays nicely with Mapzen Search, or with other open-source and proprietary geocoding options.

If you don't have the inclination to peer inside your web browser, here's what the requests look like on ours:

Status	Method	File	Domain	Type	Tran...	Size	0 ms	40.96 s	
● 200	GET	route?json={"locations":[{"lat":4...	valhalla.mapzen...	json	3.02 KB	9.59 KB			→ 432 ms
● 200	GET	reverse?point.lat=40.730608477...	search.mapzen....	json	0.64 KB	1.35 KB			→ 194 ms

Notice the speed. Mapzen Turn-by-Turn can plan journeys in hundreds of milliseconds. It's that speed that enables Mapzen Turn-by-Turn to power responsive applications. For example, **Remix** (<https://www.getremix.com>) uses Mapzen Turn-by-Turn to offer **near-instant drag-and-drop functionality to the transit planners who use its software to design and improve the arrangement of bus routes** (<https://youtu.be/o-JpgoUKr5I?t=24s>).

The Mapzen Turn-by-Turn API offers **many more options** (</documentation/turn-by-turn/api-reference/>) than we could squeeze into that nice drag-and-drop demo. Try modifying other parameters, like the departure date and time, using our test tool in these other cities:

- **Atlanta**
(<http://valhalla.github.io/demos/routing/multimodal.html#loc=14,33.776222,-84.382768&locations=%5B%7B%22lat%22:33.638632581%22%22lon%22:-84.382768%22%22%7D%5D&date=2017-12-29T12:29:00Z>)
- **Philadelphia**
(<http://valhalla.github.io/demos/routing/multimodal.html#loc=13,39.966135,-75.221071&locations=%5B%7B%22lat%22:39.995533818%22%22lon%22:-75.221071%22%22%7D%5D&date=2017-12-29T12:29:00Z>)
- **Toronto**
(<http://valhalla.github.io/demos/routing/multimodal.html#loc=15,43.665047,-79.381864&locations=%5B%7B%22lat%22:43.664891407%22%22lon%22:-79.381864%22%22%7D%5D&date=2017-12-29T12:29:00Z>)
- **Boston**
(<http://valhalla.github.io/demos/routing/multimodal.html#loc=14,42.358417,-71.057982&locations=%5B%7B%22lat%22:42.373066332%22%22lon%22:-71.057982%22%22%7D%5D&date=2017-12-29T12:29:00Z>)

The Landscape of Transit: Island, Continent, or Archipelago?

These demos may make you feel like a sea captain landing in uncharted territory. Whether it's an island or the tip of a new continent is unclear. This sense of disorientation is, in fact, progress. Previously every transit-routing engine was an island: You or your technically sophisticated friend had to fetch individual GTFS feeds from transit-agency websites and manually merge and disambiguate stop locations included in multiple feeds. The extent of the territory was always known—and always small. Now, Transitland contributors stitch together coverage for new regions every day, and we're starting to form an archipelago of interconnected transit islands:



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

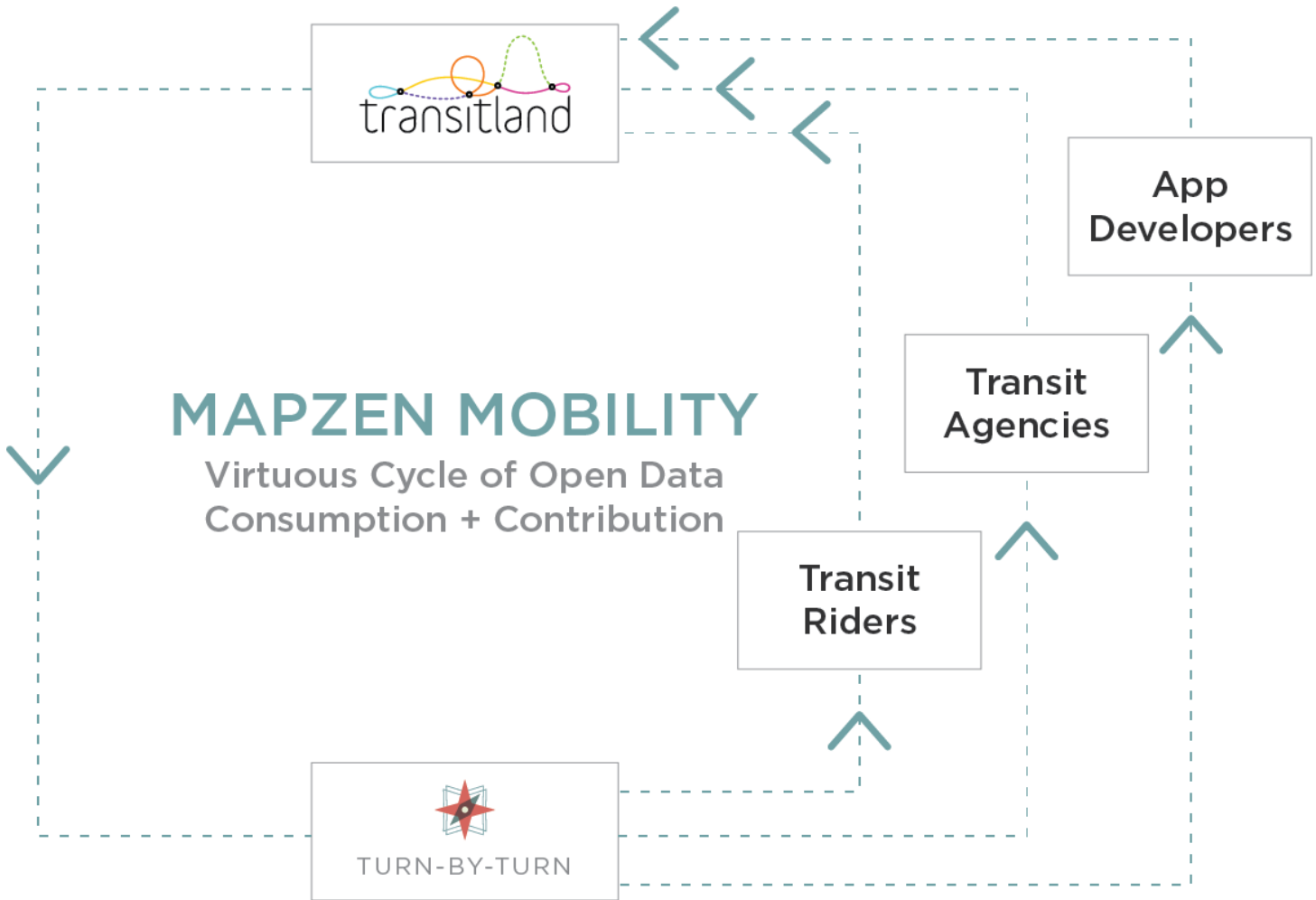
*This map shows tile squares where Mapzen Turn-by-Turn provides transit routing, using data provided by the Transitland Datastore. Each "island" of connectivity is displayed in a different color. Mapzen Turn-by-Turn uses the same methodology to create **a connectivity map of where people can travel by auto, foot, and bike according to OpenStreetMap** (</blog/you-me-and-connectivity/>). (Open the world full screen ↗ (<https://tangrams.github.io/tangram-frame?url=https://gist.githubusercontent.com/anonymous/e95849b3fbb926da23e4ff95e85295f9/raw/466fb715ab80d5afab31c94564d7cc775050a6af/scene.yaml#3.75/31.02/-53.98>))*

Are you aware of an open transit data feed that's not yet represented on that map? **Contribute it to Transitland** (</blog/help-us-catalog-the-transit-feeds-of-the-world/>) and it will appear in Mapzen Turn-by-Turn and on the transit connectivity map shortly.

North American transit enthusiasts may be disheartened to notice that "BosWash" is two separate island on the map above. The corridor between Boston and Washington, D.C. is known as the best connected region in the United States, after all, and it's served by the only American high-speed rail line. However, open data has a different landscape than proprietary and closed data sources. The mix of public, private, and quasi-public transit operators that serve "BosWash" have yet to fully release their route lines and schedules as open data sets. We hope to have more to share about open data progress soon. In the meantime, we encourage those with an interest in data licensing to **read more about how the Transitland Feed Registry helps to dispel the fear, uncertainty and doubt around transit data licenses and terms of use** (</blog/feed-registry/>). When we or our collaborators have it wrong, we welcome corrections and contributions.

Expanding the Landscape of Transit Together

That's the premise of the Mapzen Mobility: that it's possible to create a "virtuous cycle" with data consumers also contributing back to increase coverage and accuracy for themselves and others:



Toward that goal of sharing the cost and the promise of open transit data, we encourage you to browse **the Feed Registry list of 200+ "routable" places** (https://transit.land/feed-registry?import_level=4), try planning multimodal journeys using Mapzen Turn-by-Turn, and contribute additional coverage.

Already see an operator listed as "routable" in the Feed Registry but having problems planning a trip in their coverage area? Let us know by **e-mail** (<mailto:hello@mapzen.com>), **Twitter** (<https://twitter.com/mapzen>), or **GitHub issue** (<https://github.com/valhalla/valhalla/issues>). To provide transit-routing around the world—with interconnected archipelagos of dependable, high-speed transit accessible to all—is a major undertaking. With Transitland, Mapzen Turn-by-Turn, and the other Mapzen Mobility services, we're working toward that goal together with some motivated and talented partners and collaborators. Join us.

· 07 June 2016 ·



Drew Dara-Abrams

Drew leads Mapzen Mobility products (and aspires to being a flâneur).



Kristen DiLuca

Kristen is a software engineer specializing in our routing API services. She also enjoys dabbling in javascript for our open source routing test tools and always welcomes new challenges.

Duane Gearhart



Duane is a software engineer @mapzen specializing in quality route guidance and real-world route analysis.

**Meghan Hade**

Meghan is a software engineer with a background in urban planning on Mapzen's Mobility team. She works in javascript and plays in calligraphy, block printing, and finding ways to stash n+1 bikes in a San Francisco apartment.

**Greg Knisely**

Greg is a data munger and software engineer for Mapzen's routing software.

**Kevin Kreiser**

Kevin works on routing at Mapzen but secretly tries to work in all mapping disciplines. Er iss aa Pennsilfaanisch Deitscher.

**David Nesbitt**

Dave leads Mapzen Mobility engineering. Rides a variety of 2 wheel vehicles.

**John Oram**

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

**Ian Rees**

Ian spends too much time thinking about cities, land use, and open transit data on the Mapzen Mobility team.

**Alex Smith**

Opening Seattle

osm (/tag/osm)

It is with great excitement that we once again recognize the achievements of OpenStreetMap, the community that creates it and the impact OpenStreetMap has on the foundations of mapping. On July 23rd-25th the annual **State of the Map US** (<http://stateofthemap.us/>) heads to Seattle and we will be there. With so many others committed to the sustainability of OpenStreetMap we will be there sponsoring, speaking, board membering, organizing and celebrating.



Everything we do at Mapzen is open – built on open data and open source. This is because everything we do is bigger than the latest technical announcement. The collaborative community of open ensures that everything we do is sustainable beyond us. And to make a map sustainable beyond our own border, we often rely on OpenStreetMap, the only open world spatial database.

Come look for us to learn how you can improve and use OpenStreetMap to tackle geo problems and innovate geo solutions. We will have a table with experts on hand and we will be speaking at the following sessions:

- **Beyond Aesthetic Icing: Designing geo tools for humans**
(<http://stateofthemap.us/2016/beyond-aesthetic-icing/>) (Sat, 4:45) - *Ekta Daryanani, Meghan Hade*
- **Behind the Scenes of the Mapzen Targeted Editing Series - Engaging Editors**
(<http://stateofthemap.us/2016/behind-the-scenes-of-the-mapzen-targeted-editing-series-engaging-editors/>) (Sun, 4:30) - *Indy Hurt*
- **Befriending a Geocoder** (<http://stateofthemap.us/2016/befriending-a-geocoder/>)
(Sun, 11:45) - *Diana Shkolnikov*
- **Give your vector tile life** (<http://stateofthemap.us/2016/give-your-tile-life/>) (lightning talk, Sat, 4:45) - *Hanbyul Jo*
- **The Space Between: expanding street centerlines to street polygons**
(<http://stateofthemap.us/2016/project-lead/>) (lightning talk, Sat, 4:45) - *Lou Huang*
- **Make maps more interactive with the magic of a geocoding search box**
(<http://stateofthemap.us/workshops/>) (workshop, Mon 9AM) - *Rhonda Glennon, Katie Kowalsky, Diana Shkolnikov*

We are excited to once again make the annual US conference a success and we invite you to come find us to make mapping sustainable for all.

This post was updated on July 21, 2016 to add links to the abstracts for the lightning talks and workshop and update the workshop author.

· 09 June 2016 ·



Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.

The Assault on Copenhagen

[search \(/tag/search\)](#) [geocoding \(/tag/geocoding\)](#)

Second Northern War, 1655–1660

Winter was coming. Denmark and Sweden had been at war for years, along with the Dutch, Russians, Prussians, Brandenburgs, Poles, and Lithuanians.



Skirmish with Polish Tatars

Lemke, via Wikipedia

(https://commons.wikimedia.org/wiki/File:Lemke_Skirmish_with_Polish_Tatars.png)

No love was lost between the two nations. A generation prior, Sweden had been part of the Danish Empire. But in June of 1657, with King Charles X of Sweden bogged down in Poland, Denmark attacked mainland Sweden and its outlying possessions to regain lost territory.

Assault on Copenhagen



King Charles X Gustave of Sweden via Wikipedia
(https://en.wikipedia.org/wiki/Charles_X_Gustav_of_Sweden)

King Charles responded immediately. But rather than counterattacking Danish troops on the Swedish mainland, he withdrew from the years of battles in Poland and attacked westward from Swedish territory in Pomerania. His troops retook Bremen-Verden from the Danes and occupied the Danish mainland of Jutland within a month. As the Swedes sieged Danish forts that autumn, the Danes strategically withdrew to their island stronghold of Copenhagen for the winter, not expecting an attack until spring.



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

King Charles X's advance on Copenhagen, 1657-58

Danish territory in red, Swedish territory in blue.

Yellow/blue stripes: Danish provinces lost in 1658 after the Treaty of Roskilde.

Red/blue stripes: Norwegian provinces lost by Denmark in 1658 but regained by 1860.

Click to occupy Copenhagen in full screen ↗ (<https://tangrams.github.io/tangram-frame/?noscroll&url=https://mapzen-assets.s3.amazonaws.com/resources/assault-on-copenhagen.yaml#7/54.898724/11.239014>)

Unfortunately for the Danes, the winter was extraordinarily cold, and the straits between Jutland and the islands leading to Copenhagen froze over. On January 30 of 1658, King Charles X sent 12,000 soldiers marching over the frozen “Little Belt” with no losses. They then island-hopped over the “Big Belt”.



Crossing of the Belts

by Lemke, via Wikipedia (https://en.wikipedia.org/wiki/March_Across_the_Belts)

In a panic, Denmark signed a peace treaty on February 18, and by February 26, the Treaty of Roskilde where they gave up their remaining territory on the Swedish peninsula (the provinces of Scania, Halland, Bohusl nd and Bleckinge). The Danish Empire was starting to fade.

Despite the treaty, King Charles attacked Denmark **again** in the summer of 1658, but the Dutch sent a fleet to provide supplies and assistance and defeated the Swedish navy within sight of Copenhagen (and the newly acquired Swedish coastline).



Danish, Dutch and Swedish fleets, summer of 1658 via Wikipedia ([https://en.wikipedia.org/wiki/Dano-Swedish_War_\(1658%E2%80%931660\)#/media/File:K%C3%B8benhavn_s_storm_1659.jpg](https://en.wikipedia.org/wiki/Dano-Swedish_War_(1658%E2%80%931660)#/media/File:K%C3%B8benhavn_s_storm_1659.jpg))

But wait, there's more. King Charles attacked Copenhagen yet **AGAIN** in February of 1659, but this time the Danes were prepared.



Assault on Copenhagen (1659) during the 1658/59 siege

Frederik Christian Lund, via Wikipedia

(https://en.wikipedia.org/wiki/Second_Northern_War#/media/File:Stormningen_av_K%C3%B6penhamn_11_feb._1659.jpg)

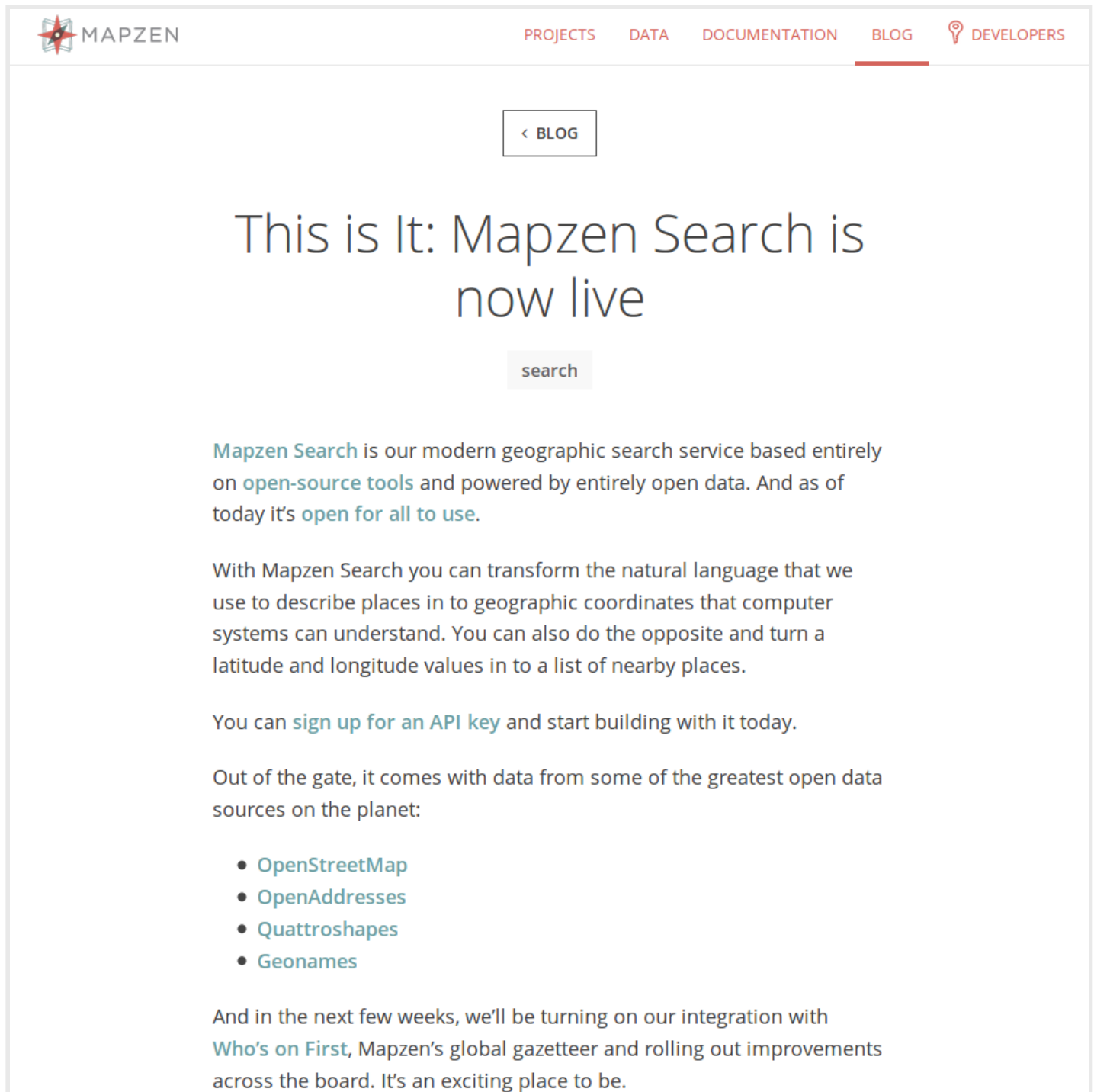
They cut holes in the ice to prevent another military crossing, and the remaining Swedish troops were forced to pull back to the Danish mainland. That summer, Polish, Austrian and Brandeburger troops defeated the Swedish Army in Jutland and thousands of Swedish troops were taken prisoner.

King Charles X died in February of 1660. Soon after, Denmark and Sweden signed the Treaty of Copenhagen, "**formalizing** (https://en.wikipedia.org/wiki/Dano-Swedish_war)" the borders between Denmark, Sweden and Norway.

FORESHADOWING: Denmark was able to keep the island of Bornholm in exchange for giving up territory in southern Sweden.

2016

In April 2016, the Pelias Geocoder team at Mapzen had just pushed a major update with vast improvements in both the code and the data.



The screenshot shows the Mapzen website's blog page. At the top, the Mapzen logo is on the left, and navigation links for PROJECTS, DATA, DOCUMENTATION, BLOG (which is highlighted with a red underline), and DEVELOPERS are on the right. Below the navigation, a button labeled '< BLOG' is centered. The main heading reads 'This is It: Mapzen Search is now live'. Below the heading is a search input field with the placeholder text 'search'. The body text describes Mapzen Search as a modern geographic search service based on open-source tools and open data. It lists sources like OpenStreetMap, OpenAddresses, Quattroshapes, and Geonames. The post concludes by mentioning upcoming integrations with Who's on First.

MAPZEN

PROJECTS DATA DOCUMENTATION **BLOG** DEVELOPERS

< BLOG

This is It: Mapzen Search is now live

search

Mapzen Search is our modern geographic search service based entirely on [open-source tools](#) and powered by entirely open data. And as of today it's [open for all to use](#).

With Mapzen Search you can transform the natural language that we use to describe places in to geographic coordinates that computer systems can understand. You can also do the opposite and turn a latitude and longitude values in to a list of nearby places.

You can [sign up for an API key](#) and start building with it today.

Out of the gate, it comes with data from some of the greatest open data sources on the planet:

- [OpenStreetMap](#)
- [OpenAddresses](#)
- [Quattroshapes](#)
- [Geonames](#)

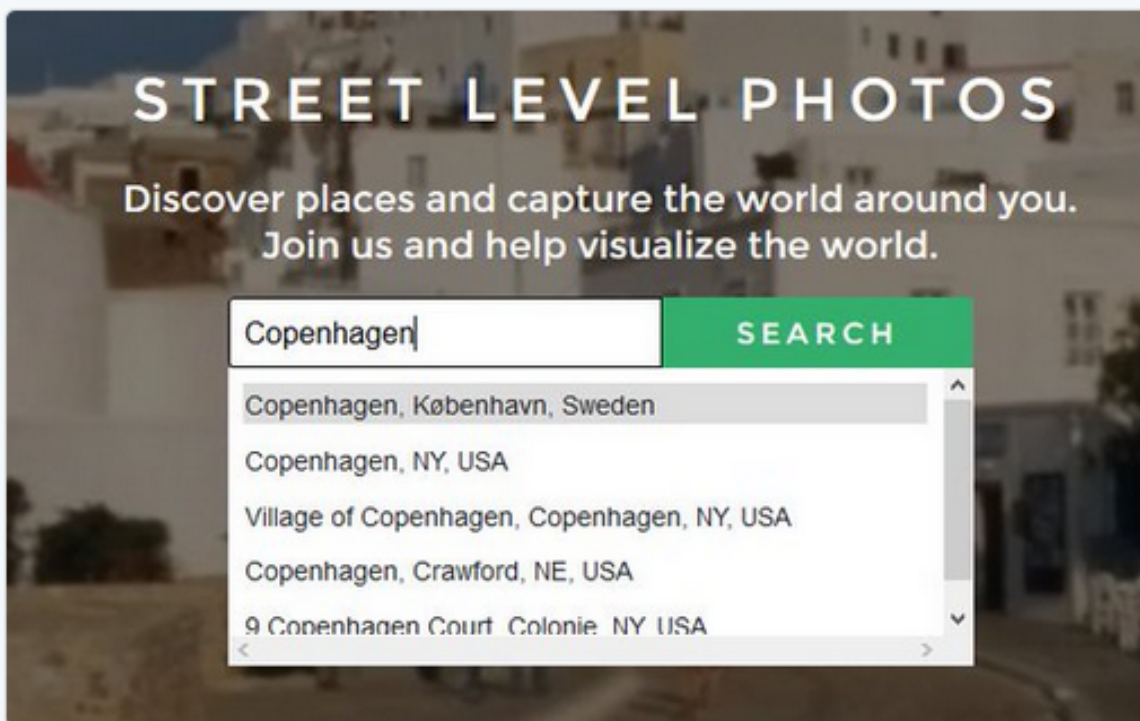
And in the next few weeks, we'll be turning on our integration with [Who's on First](#), Mapzen's global gazetteer and rolling out improvements across the board. It's an exciting place to be.

Everything looked good until reports suddenly started coming in: According to **users of Mapzen Search** (<http://blog.mapillary.com/update/2015/10/15/powered-by-mapzen-search.html>), Copenhagen was part of... Sweden! Had King Charles X finally occupied Copenhagen three and a half centuries later?



Peter Brodersen @peterbrodersen · Apr 21

Dear @mapillary. You are playing with fire.



International tensions were high, and the Pelias and Who's On First teams raced to diagnose the problem.



Jan Erik Solem

@jesolem



Follow

@peterbrodersen @mapillary Indeed!
@mapzen Search needs to fix that before the riots start.

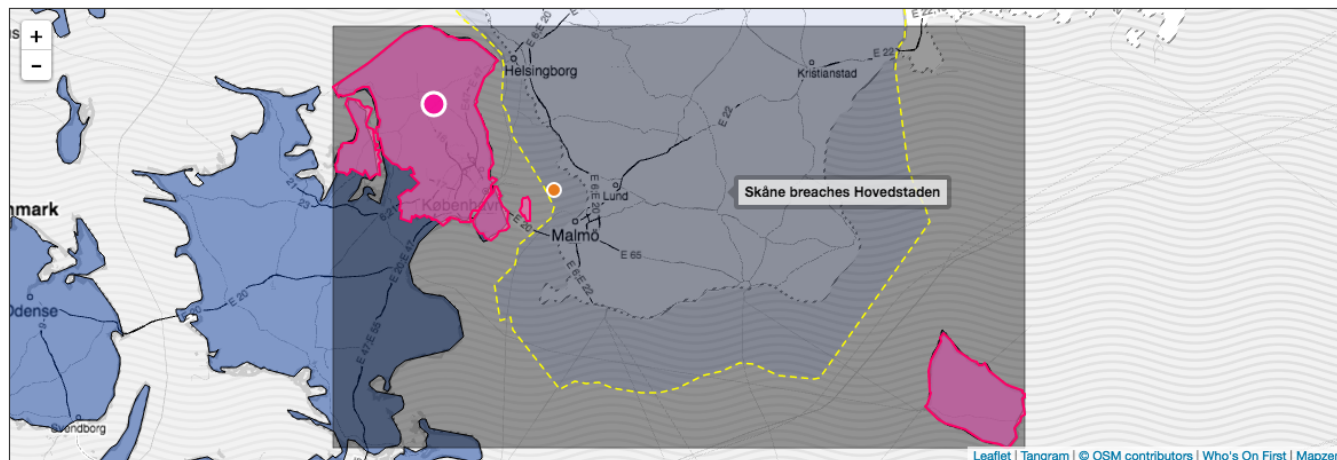
12:08 AM - 22 Apr 2016



Like any geocoder, Pelias has to be concerned with not just where places are but what other places they belong to. We quickly determined that the region of Denmark surrounding Copenhagen, known as Hovedstaden, was **also** being reported as part of Sweden.

Hovedstaden (which basically means `capital city area`) has an unusual shape. It consists of the area immediately around Copenhagen, as one expects, but also an island far to the east — wait for it... *BORNHOLM*.

Hovedstaden 856/825/81/85682581.geojson



Hovedstaden is a `region` and its consensus geometry is derived from [quattroshapes](#). Its `label` centroid is derived from [mapshaper](#). Take a [screenshot of this map](#) (this may require a few seconds to complete)

Properties — some notes about [sources](#) and [names](#)

[view raw](#)

edtf

cessation	uuuu
inception	uuuu

geom

Hierarchy

the `region` of [Hovedstaden](#)
 the `continent` of [Europe](#)
 the `country` of [Denmark](#)

Other

[See all the descendants of Hovedstaden](#)
[Raw data \(GeoJSON\)](#)

Directly in between Copenhagen and Bornholm? The southern tip of Sweden, once known as Scania, which once was part of... Denmark.

The country parentage of Hovedstaden was originally calculated in our data by determining the “centroid” of Hovedstaden, and determining which country that point was in.

But which centroid? That’s an interesting question. In **Who’s On First** (<https://whosonfirst.mapzen.com/>), we generate multiple centroids for places. The `geom` centroid is effectively the geographic center, generated using Sean Gilles’ **Shapley** (<http://toblerity.org/shapely/manual.html>). The `lbl` centroid, is more of like a... center of mass. It’s a spot where you’d probably want to put a label, and is generated using Matthew Block’s **Mapshaper** (<https://github.com/mbloch/mapshaper>). Taking a look at the map of Hovedstaden, the big pink dot on the map above is the `lbl` centroid, and the little orange dot

is the geom centroid... (**See also San Francisco** (<https://whosonfirst.mapzen.com/spelunker/id/85922583/#10/37.7850/-122.7278>) for another troublesome island centroid issue.)

While we **fixed the parentage of Hovedstaden** (<https://github.com/whosonfirst/whosonfirst-data/issues/255>), the Pelias team members are students of history and have battled bugs before and we knew there was more to do. We **wrote a solid test case** (<https://github.com/pelias/fuzzy-tests/pull/32>) for Copenhagen and all the world's capital cities. Also, Who's on First **can now detect** (<https://github.com/whosonfirst/whosonfirst-data/pull/254>) when other countries breach bounding boxes (as indicated by the yellow dotted line and hover-over warning **in the Hovedstaden map** (<https://whosonfirst.mapzen.com/spelunker/id/85682581/#8/55.563/13.497>)). And **we have tools** (<https://github.com/whosonfirst/go-whosonfirst-pip/>) for reversegeo centroids, a.k.a. point-in-polygon.

These changes, like the city walls of old, will hopefully keep King Charles out of Copenhagen.

· 21 June 2016 ·



Julian Simioni

Writer of codes at @Mapzen. Eater of ramen, rider of bikes, flyer of planes.



John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.



Aaron Straup Cope

Aaron is happiest in the presence of olive oil.

Maps Camp at the United Nations

osm (/tag/osm)

The United Nations. Home of world cooperation, respect, peace... and open maps!! So we are heading there again! Last year we hosted State of the Map US there, and the Summer of 2016 has us co-hosting the largest ever open community conference the world has ever seen. So of course, we'll be there. And we want you there too!



Come to Maps Camp at the United Nations. (photo by @burritojustice)

We welcome you to **Maps Camp** (<http://mapscamp.io/>) on July 9th, 2016 at the United Nations. We'll be part of the mappers contingency as **OpenCamps** (<http://opencamps.org/>), a series of 30+ camps that bring together 6,000+ attendees excited about open source communities. Along with Mapbox, Esri, CartoDB, and OpenStreetMap US we are thrilled to be a part of this event.

Maps Camp will be a full day of talks, panels, and workshops with leaders from a range of open mapping communities. From government agencies to the Humanitarian OpenStreetMap team, from Mapillary to Colorado visitors, we'll all be there to discuss mapping in open source. We'll talk the history of GPS (yay!), open philosophies and how you really make community. There will also be representatives from Mapzen on hand to answer any questions you have about our tools. Ohh, and food.

Interested in giving a lightning talk about what you're working on? There will be time for that, too! Whether you're new to mapping or a seasoned geospatial professional, come on over – register at the Maps Camp **website** (<http://mapscamp.io/>). You can also talk to us in 140 character or less (or more) on **@maps_camp** (https://twitter.com/maps_camp), via email at **info@mapscamp.io** (<mailto:info@mapscamp.io>), or through the Maps Camp **GitHub repo** (<https://github.com/MapsCamp>). We hope to see you there!

· 23 June 2016 ·



Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.

© 2017 Mapzen

Updating Neighbourhood Records in Who's on First

whosonfirst (/tag/whosonfirst) **tutorial** (/tag/tutorial) **data** (/tag/data)

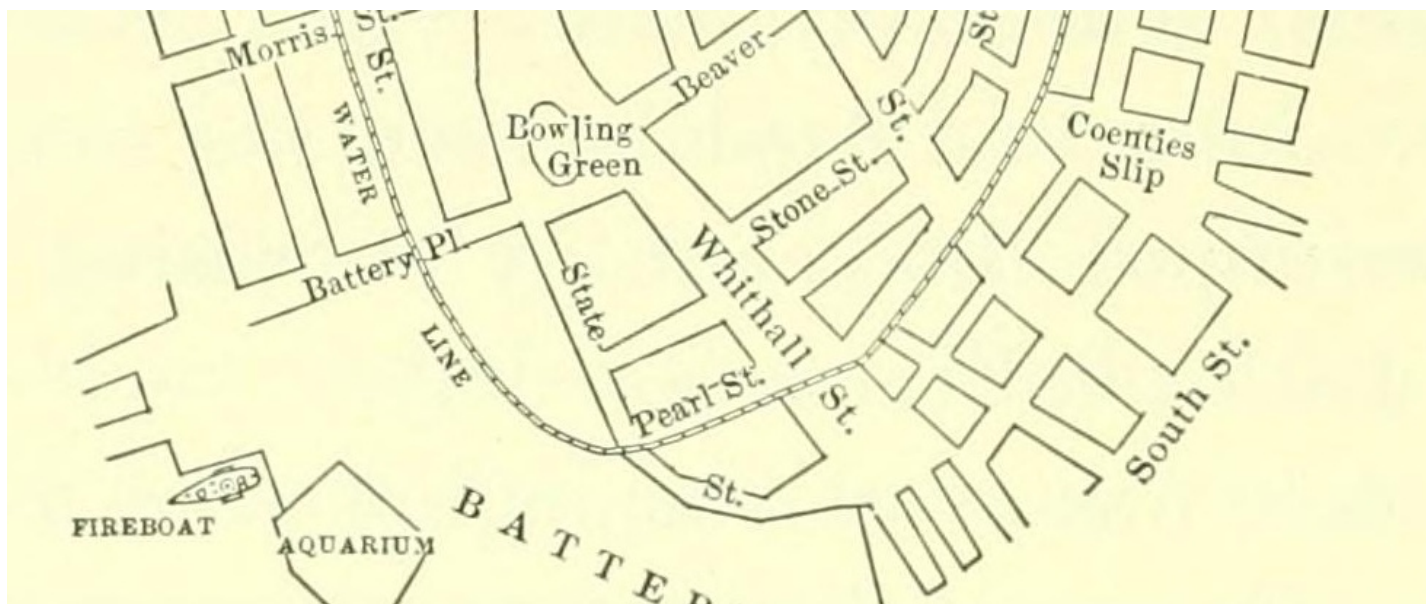


Photo Credit: **Gustave Straubenmuller, Flickr**

(<https://www.flickr.com/photos/internetarchivebookimages/14782952795/in/photolist-owjuQt-oeXUZF-oskqGG-6cfLKf-owddJm-owqudn-owtHpZ-owEBqR-owjvdn-ou63Tn-ov9vB1-ownwLy-osPEQ5-oddXQL-ovv6CR-oePvaL-odFukR-ounhb6-odFyjT-5ZzStu-4ibzJo-ouq595-x4qKZf-ouGC9w-oeY2hR-owg7Gq-odeqaE-owrPU6-oskoPo-ouooHD-odqeuY-ouq6SW-oweTDU-owSur6-oy5fzP-i7JBGv-oeX74i-ouAb8q-owq57g-y8iw81-ouMDmm-zhvmPt-Dczajt-w9YJux-oeXeWH-xnwY2E-owaNsr-oeWTFN-owrSwv-oycF32>)

Here at Mapzen, we *love* neighbourhoods - you've seen them on our maps and in your search results. We acknowledge there are issues with our data and hope that you'll use your local knowledge and expertise to help update our neighbourhood records in Who's on First. We want your help!

The rundown:

- We have a Gazetteer called **Who's on First**
- We use it for, well, **everything**
- We love neighbourhoods, but some of our neighbourhoods **could be improved**

- We **need your help** to improve them
- We have a **tutorial and list of steps** (https://github.com/whosonfirst/whosonfirst-cookbook/blob/master/issue_workflows/sf_neighbourhood_updates_pt_1.md) for you to follow if you'd like to contribute

We have a Gazetteer called Who's on First

A while back¹, we wrote about the **Who's On First** (<https://whosonfirst.mapzen.com/>) project. Who's on First is a gazetteer of places... not quite all the places in the world, but a whole lot of them and, we hope, the kinds of places that we mostly share in common.

*A **gazetteer** is a big list of places, each with a stable identifier and some number of descriptive properties about that location.*

It is essentially a (very) large collection of open administrative and venue data. We've developed a useful browsing tool that displays our Who's on First records' geometries and attributes in our **Spelunker** (<https://whosonfirst.mapzen.com/spelunker/>).

We use it for, well, everything

Who's on First is used in many Mapzen services. Specifically, neighbourhood records in Who's on First are important because they are used for:

- **Labelling:** When we use a mapping application, we want neighbourhoods to be labelled on the map (in the right place and at the right zoom)
- **Ability to search:** Neighbourhoods should be searchable (with the ability to know how large the feature is and fit it into view)
- **Browsing venues:** We should be able to browse venues by neighbourhood

We love neighbourhoods (some of our neighbourhoods could be improved)

Not every gazetteer includes neighbourhood geometries, but Who's On First *does* and we're proud of that. We've spent a lot of time cleaning up their names and positions on the map, and now we're starting to clean up their shapes.

While Who's On First uses **Quattroshapes (www.quattroshapes.com)** geometries for most neighbourhoods globally, many neighbourhoods in the United States, including San Francisco, source their default geometry from **Zetashapes (www.zetashapes.com)**. The Zetashapes project follows the same basic principles as Quattroshapes, but builds shapes up from Census 2010 features. We've seen problems with shapes extending into adjacent localities, counties, and sometime far out into neighboring rural areas. This automated technique used by Zetashapes is responsible for some of the issues that we want to address. We want shapes touched by a human, and we'll **show you how to do so in San Francisco** (https://github.com/whosonfirst/whosonfirst-cookbook/blob/master/issue_workflows/sf_neighbourhood_updates_pt_1.md) as this map illustrates:



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

San Francisco macrohoods (red), WOF neighbourhoods (blue), additional SFGov neighbourhoods (green) & microhoods (purple).

Click to view San Francisco in full screen ↗ (https://tangrams.github.io/tangram-frame/?lib=0.8&noscroll&url=https://s3.amazonaws.com/whosonfirst.mapzen.com/misc/_blogs/neighbourhood_blogpost.yaml#13/37.7669/-122.4398)

We need your help to improve them

Because we (unfortunately) do not have expert knowledge of every neighbourhood on Earth, we are hoping you'll help us improve these neighbourhood shapes for your locality!

Big cities love neighbourhoods. Neighbourhoods like *Kensington Gardens (London)*, *Capitol Hill (Seattle)*, *The Mission (San Francisco)*, and *French Quarter (New Orleans)* are well-known and often mentioned in apartment rental and housing ads. Smaller cities might have a couple well known neighbourhoods, but they're "squishy", and you'd probably say things like "by the shopping mall" or "by the high school" instead of using a neighbourhood name.

In larger localities, like San Francisco and New York City, we generate neighbourhood names in our **Search** (<https://mapzen.com/projects/search/>) results for the more interesting neighbourhoods. This is why we want our neighbourhoods to be as clean and as accurate as possible - your local knowledge and expertise will help us accomplish this.

We have a tutorial and list of steps for you to follow if you'd like to contribute

If this sounds like a project you'd like to contribute to, check out our **tutorial on updating Who's on First records** (https://github.com/whosonfirst/whosonfirst-cookbook/blob/master/issue_workflows/sf_neighbourhood_updates_pt_1.md).

We look forward to hearing from you!

-
1. Blog posts **introducing Who's On First** (<https://mapzen.com/blog/who-s-on-first>), using the Spelunker and **Jumping into Who's On First** (<https://mapzen.com/blog/spelunker-jumping-into-who-s-on-first>), WOF neighbourhoods in **Mapzen Vector Tiles** (<https://mapzen.com/blog/vector-tiles-v0-8-preview>), and WOF places powering **Mapzen Search** (<https://mapzen.com/blog/wof-search-live/>) ↪

· 24 June 2016 ·



Stephen Epps

Stephen is Mapzen's gardener and fixer-upper of geographic data.

© 2017 Mapzen

Voice Guidance

routing (/tag/routing)

It matters! Having a voice and choosing the right words matter. Whether it is the Voice that gets the popular vote on the acclaimed TV show or the words that capture the attention of voters on a heated political debate — it matters! It is no different in the mapping industry. Today, open mobile navigation has a voice, and choosing the right words for the turn-by-turn service matters.

Verbal Instructions

The Mapzen Turn-by-Turn service is mobile focused, and it is essential to have verbal instructions that will work well with various text-to-speech(TTS) engines. Depending on the maneuver, we return textual instructions as well as up to three verbal instructions. These include the following:

- The **verbal transition alert instruction** is intended to provide advance notice of an impending maneuver – kind of a wake up call to get the user prepared to make a maneuver.
- The **verbal pre transition instruction** is more detailed and is presented just before the maneuver is made.
- The **verbal post transition instruction** describes what to do after making the maneuver - usually something like continue for some distance, but can also identify if the road name changes.

Maneuver Instructions Example:

Maneuver Item	Value
instruction	Turn right onto East Chocolate Avenue/US 422. Continue on US 422.
verbal_transition_alert_instruction	Turn right onto East Chocolate Avenue.

verbal_pre_transition_instruction	Turn right onto East Chocolate Avenue, U.S. 4 22.
verbal_post_transition_instruction	Continue on U.S. 4 22 for 3.3 miles.

TTS Enhancements

Another objective of this service is to provide TTS enhancements that mimic how people actually answer when giving verbal directions. We format or expand route names so they are spoken correctly. The TTS enhancements can be implemented based on location. Since the United States' road network is large and diverse, we chose this region and selected several examples from different states.

For US routes, we transform **US** to **U.S.** so it is pronounced as **U S** and not as the pronoun **us** or the expanded **United States**.

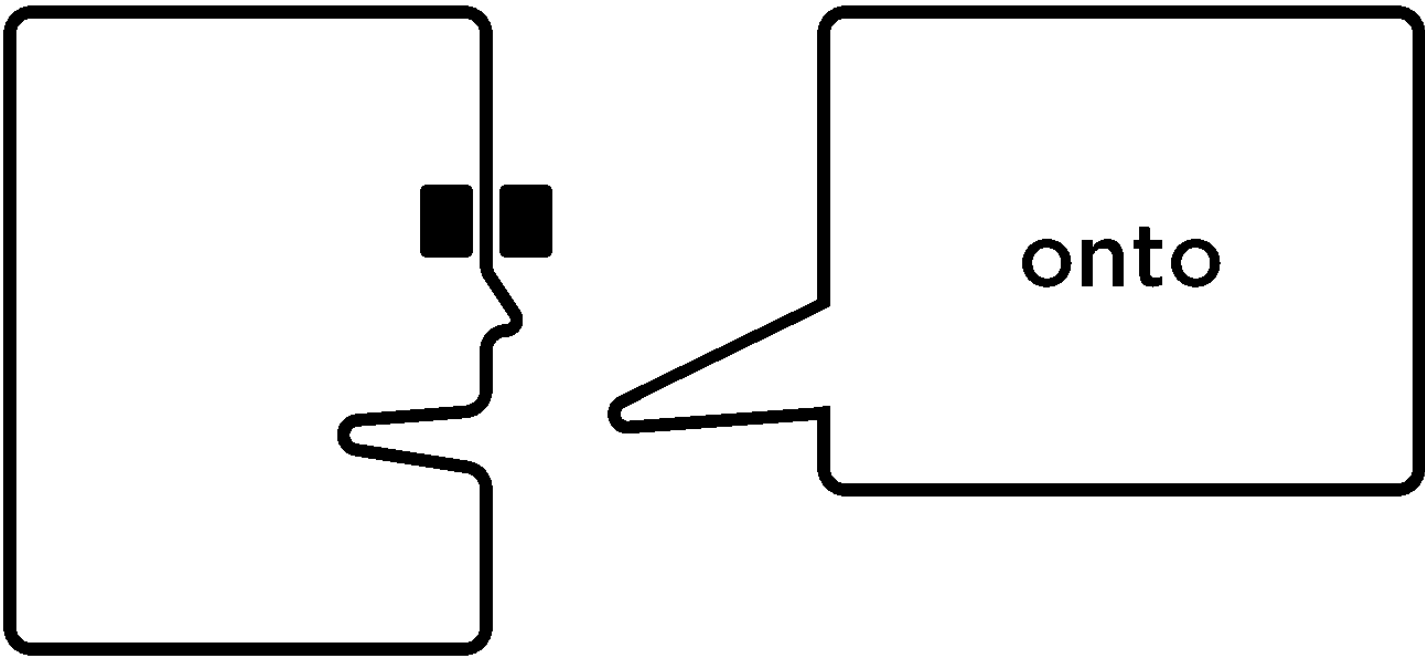
Textual	Verbal	Location
US 1 North	U.S. 1 North	United States
US 22 East	U.S. 22 East	United States

For state routes, we expand abbreviations - for the first example below, **MA 3 South** will be expanded to **Massachusetts 3 South** so it will not be pronounced as **Maw three South**.

Textual	Verbal	Location
MA 3 South	Massachusetts 3 South	United States
MD 32 West	Maryland 32 West	United States

For route numbers and house numbers, we group and format the numbers so they will be pronounced properly. For numbers greater than two digits, the numbers are transformed into

groups of two, from right to left. In the example below, the interstate number **695** is transformed to **6 95** , so it will be pronounced **six ninety-five** instead of **six hundred ninety-five** .



Similarly, the four digit state route **1021** is transformed to **10 21** , so it will be pronounced **ten twenty-one** instead of **one thousand twenty-one** . When the numbers are grouped by two, any leading zero will become the letter **o** . Therefore, the US route **101** is transformed to **1 o1** , so it will be pronounced **one oh one** instead of **one hundred one** . For any number that ends with a double zero, the double zero is transformed to **hundred** . The house number 7700 below is transformed to **77 hundred** , so it will not be pronounced as **seven thousand seven hundred** . In the final example below, notice how the house number **905** is transformed to **9 o5** , however the street name **203rd** does not need changed because it is already properly formatted for verbal directions.

Textual	Verbal	Location
I 695 West	Interstate 6 95 West	United States
SR 1021	State Route 10 21	United States
US 101 North	U.S. 1 o1 North	United States
7700 Mapzen Way	77 hundred Mapzen Way	United States
905 203rd Street	9 o5 203rd Street	United States

The examples below represent location specific interpretations. The first two examples demonstrate the difference between interpreting the route name as either a state abbreviation or as a county road. The third example represents a route specific to the state of Texas.

Textual	Verbal	Location
CO 7	Colorado 7	Colorado, United States
Co 7	County Road 7	Ohio, United States
FM 1018	Farm to Market Road 10 18	Texas, United States

Multi-cue Instructions

The Mapzen Turn-by-Turn service will detect short maneuvers and combine separate verbal instructions into one multi-cue instruction so the user will be prepared for each transition. The following example illustrates how in the 2nd maneuver we append the **verbal pre transition instruction** with the next **verbal transition alert instruction**.

2: Turn left onto North Plum Street.

VERBAL_ALERT: Turn left onto North Plum Street. 89 m

VERBAL_PRE: Turn left onto North Plum Street. Then Turn right onto East Fulton Street.

VERBAL_POST: Continue for 90 meters.

3: Turn right onto East Fulton Street.

VERBAL_ALERT: Turn right onto East Fulton Street. 107 m

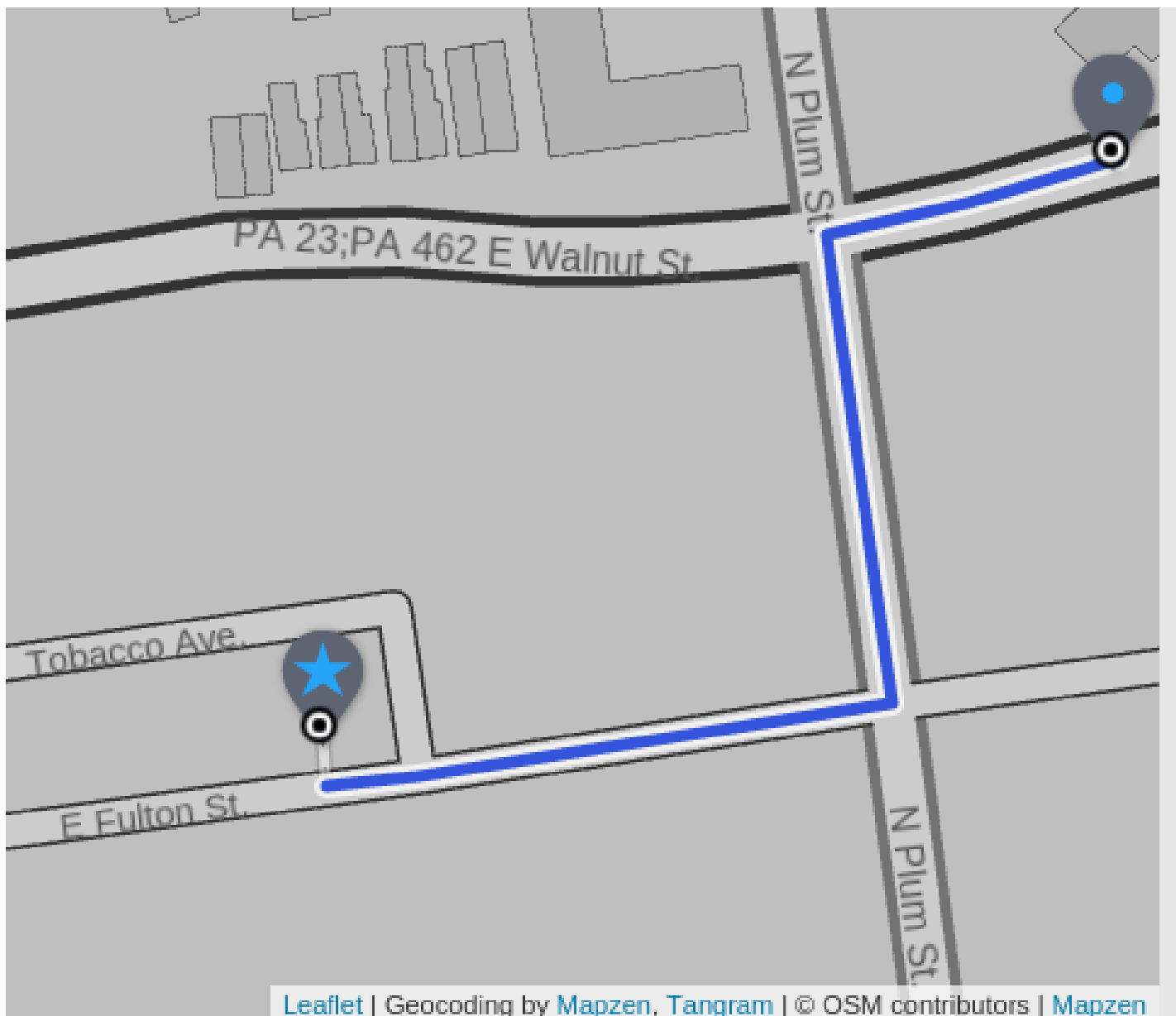
VERBAL_PRE: Turn right onto East Fulton Street.

VERBAL_POST: Continue for 100 meters.

4: Your destination is on the right.

VERBAL_ALERT: Your destination will be on the right. 0 m

VERBAL_PRE: Your destination is on the right.



Enhancements like these matter because it matters what you say and how you say it. Every turn-by-turn service has something to say – we just strive to say it better! You can hear the voice guidance in Eraser Map (<https://mapzen.com/blog/erasermap-beta>), our privacy-focused Android application.

Check out the **Mapzen Turn-by-Turn documentation** (<https://mapzen.com/documentation/turn-by-turn/>) and **let us know if you have any questions** (<https://twitter.com/intent/tweet?text=@Mapzen%20@ValhallaRouting%20Hi!>).

· 30 June 2016 ·



Duane Gearhart

Duane is a software engineer @mapzen specializing in quality route guidance and real-world route analysis.

© 2017 Mapzen

Walkabout Map Style

cartography (</tag/cartography>) **tangram** (</tag/tangram>) **vector-tiles**
(</tag/vector-tiles>)

A few months ago we asked ourselves: how could we celebrate walking? Today we introduce Mapzen's latest cartography: **Walkabout**.

Walkabout features walking paths & hiking trails *early*, emphasizes outdoor attractions with bright green-blue icons, and shows where the ups and downs are with hill shading. From a morning jog to a slow stroll with friends, in the city to the backcountry, Walkabout evokes the places we've played outdoors and inspires us to "get outside".

Walkabout joins Mapzen's other house styles: **Bubble Wrap** (<https://mapzen.com/blog/bubble-wrap-carto/>), **Refill, Zinc, and Cinnabar** (<https://mapzen.com/blog/introducing-refill-cinnabar-and-zinc-styles-for-tangram>) and is available immediately in **default** (<https://github.com/tangrams/walkabout-style>), **more labels** (<https://github.com/tangrams/walkabout-style-more-labels>), and **no labels** (<https://github.com/tangrams/walkabout-style-no-labels>) basemap versions.

Explore Walkabout below and scroll down for a visual tour of this signature cartography.

[Leaflet](#)

Walkabout San Francisco. View full screen ↗ (<https://tangrams.github.io/walkabout-style-more-labels/#12/37.7753/-122.4213>)

Icons

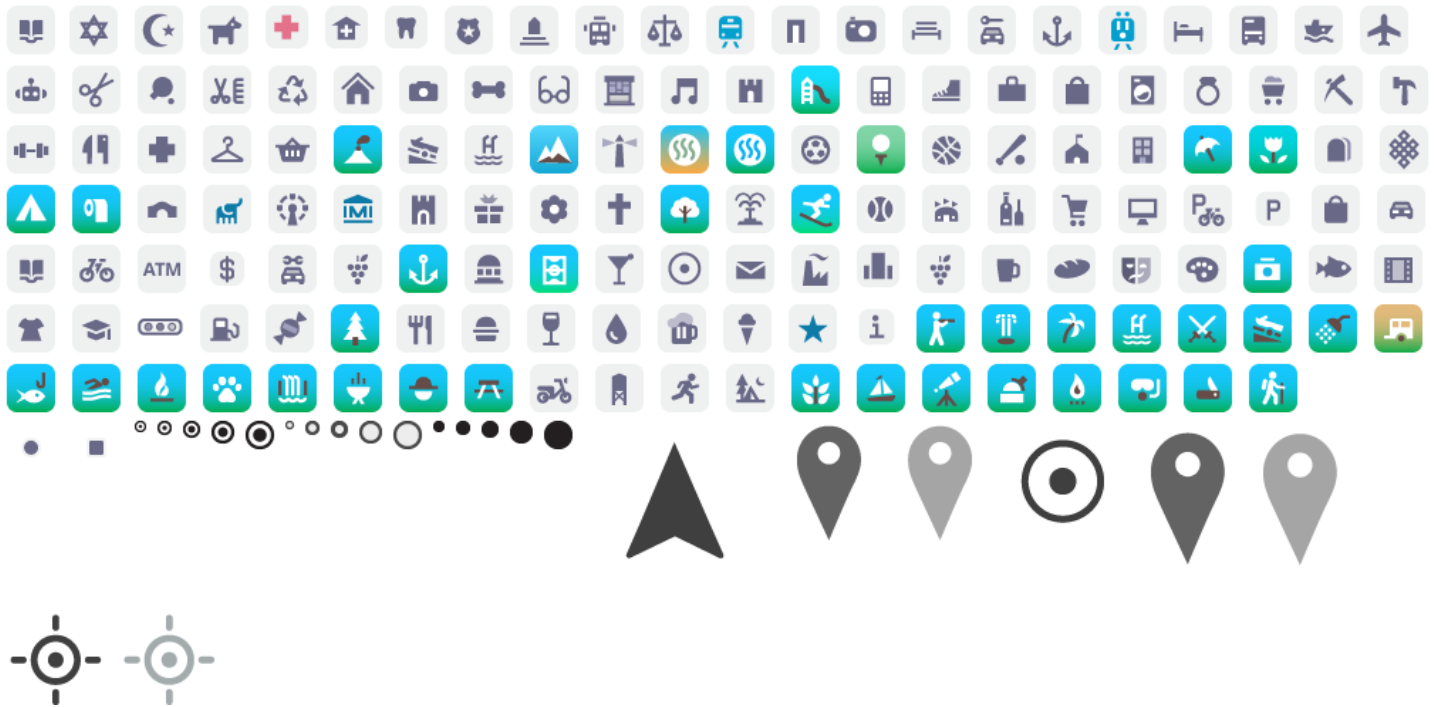
We designed a completely new set of icons for **Walkabout** focusing on crisp details in a 18 pixel square grid. Outdoor related icons are emphasized with a blue-green gradient color badge and a white glyph with dark brown highlights.

This treatment allows Walkabout icons to figure above the basemap cartography drawing attention to features like parks, beaches, ski resorts, campgrounds, picnic sites, restrooms, trailheads, peaks, waterfalls, dog parks, and outdoor supply stores.

We're not just interested in the destination, though. Sometimes it's about the journey. Or maybe you get hurt and need help. Or maybe you're more interesting in urban walking. These kinds of places feature a dash of color on a muted background and include train stations, hospitals, zoos, museums, and tourist attractions.

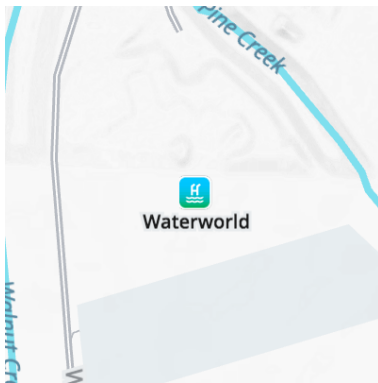
All other features are drawn in a muted grey, from restaurants to sports pitches.

Can you spot all the custom icons in each scene below?

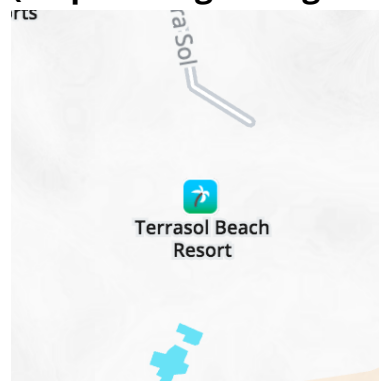


Icons on the map

All images below click thru to full screen interactive maps.

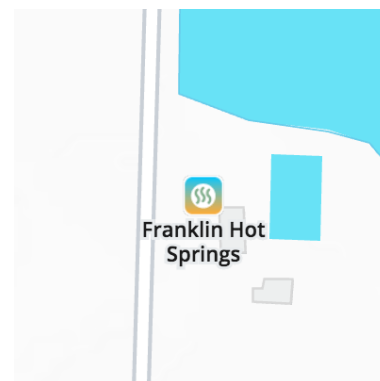


(<https://tangrams.github.io/walkabout-style-more-labels/#16/37.9740/-122.0515>)



(<https://tangrams.github.io/walkabout-style-more-labels/#17/22.87750/-109.90505>)

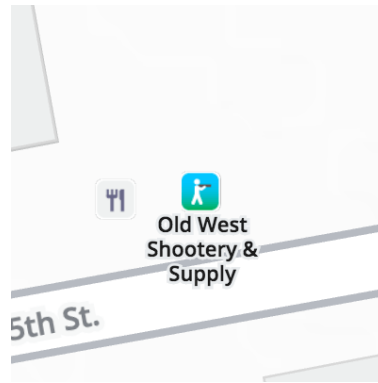
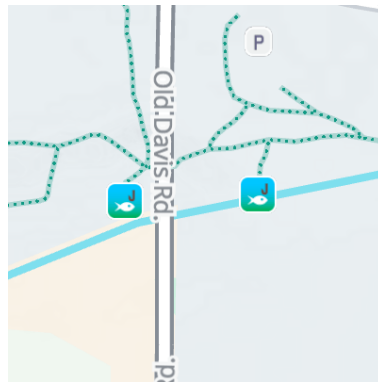
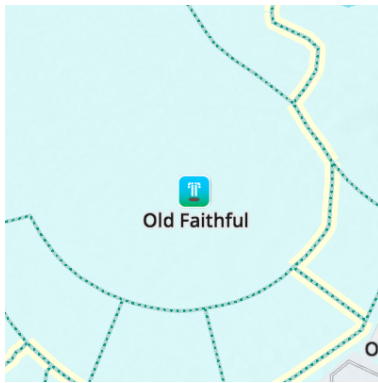
(<https://tangrams.github.io/walkabout-style-more-labels/#18/35.58797/-120.64210>)



(<https://tangrams.github.io/walkabout-style-more-labels/#16.19927/44.46034/-110.82686>)

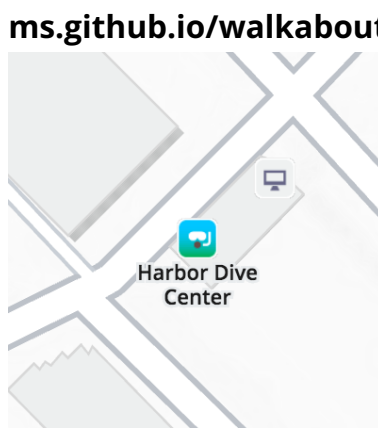
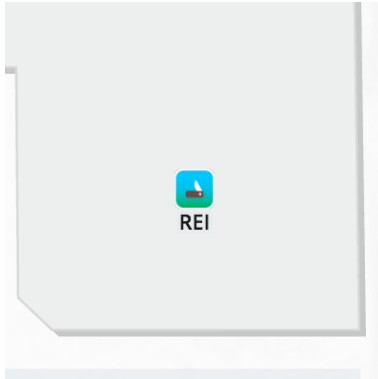
(<https://tangrams.github.io/walkabout-style-more-labels/#17/38.51730/-121.75540>)

(<https://tangrams.github.io/walkabout-style-more-labels/#17/38.51730/-121.75540>)



(<https://tangrams.github.io/walkabout-style-more-labels/#19/40.80336/-124.15660>)

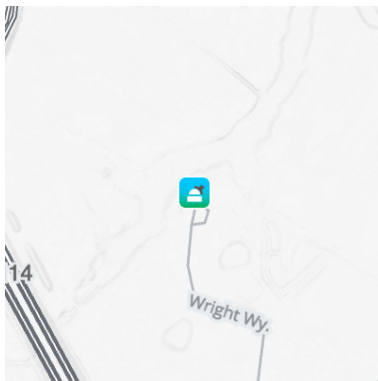
(<https://tangrams.github.io/walkabout-style-more-labels/#18/40.70072/-111.80074>)



(<https://tangrams.github.io/walkabout-style-more-labels/#18/37.86818/-122.49844>)

(<https://tangrams.github.io/walkabout-style-more-labels/#18/37.28474/-121.99900>)

(<https://tangrams.github.io/walkabout-style-more-labels/#15/39.2775/-75.5798>)



(<https://tangrams.github.io/walkabout-style-more-labels/#20/37.34121/-121.64287>)

(<https://tangrams.github.io/walkabout-style-more-labels/#18/40.19933/-88.37883>)

(<https://tangrams.github.io/walkabout-style-more-labels/#17/45.56141/-78.70565>)

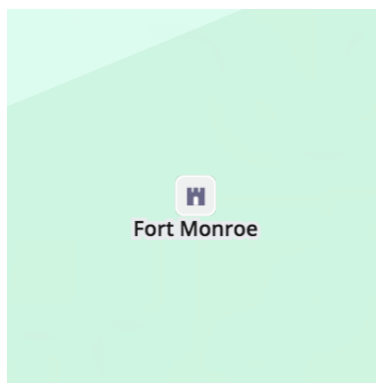
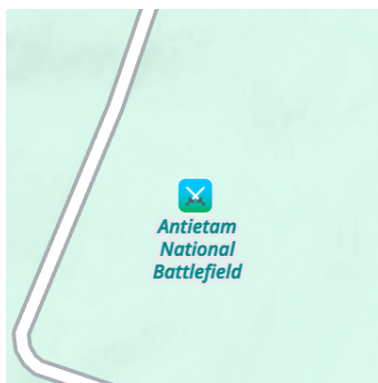
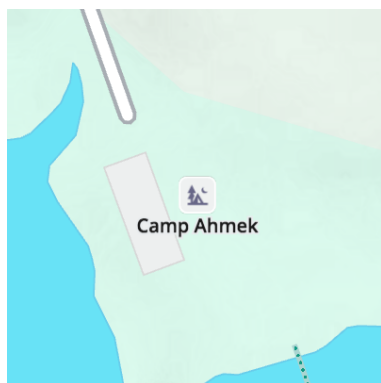
(<https://tangrams.github.io/walkabout-style-more-labels/#17/39.46753/-77.73610>)

(<https://tangrams.github.io/walkabout-style-more-labels/#20/37.00422/-76.30679>)

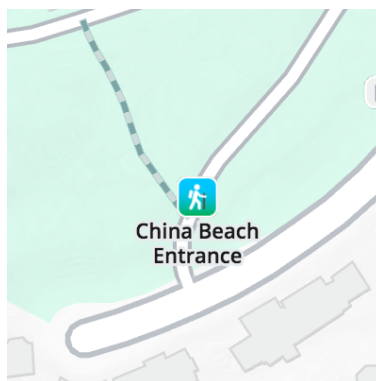
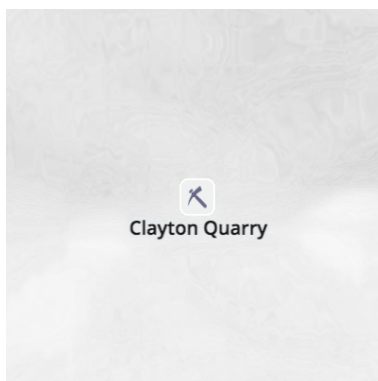
(<https://tangrams.github.io/walkabout-style-more-labels/#20/37.77386/-122.41115>)

(<https://tangrams.github.io/walkabout-style-more-labels/#17/37.92515/-121.94668>)

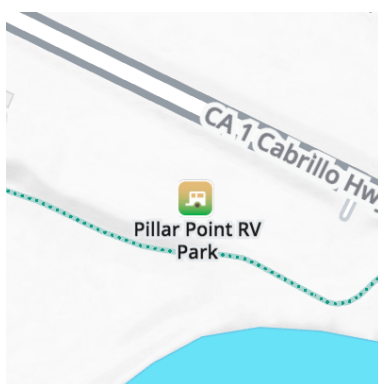
(<https://tangrams.github.io/walkabout-style-more-labels/#18/37.78743/-122.49095>)



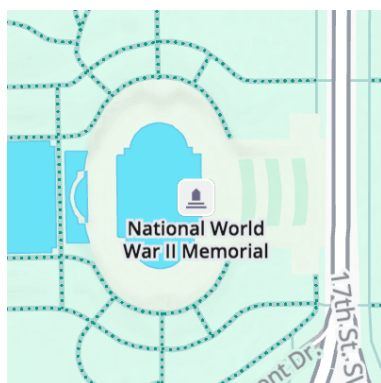
(<https://tangrams.github.io/walkabout-style-more-labels/#14/37.9043/-122.6008>)



(<https://tangrams.github.io/walkabout-style-more-labels/#17/37.50136/-122.47313>)



(<https://tangrams.github.io/walkabout-style-more-labels/#20/38.54985/-121.78582>)



(<https://tangrams.github.io/walkabout-style-more-labels/#17/38.88952/-77.04016>)

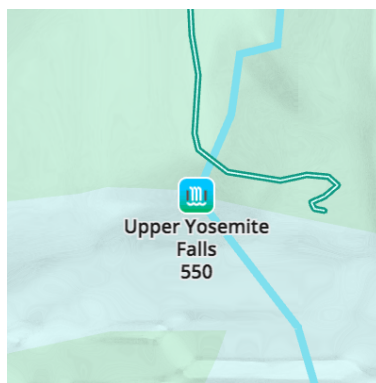
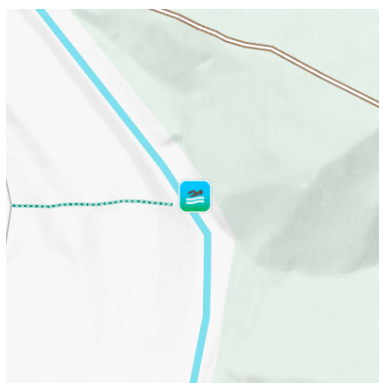
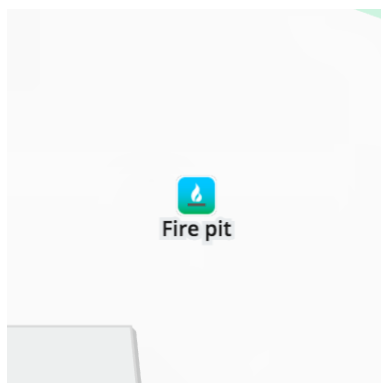
(<https://tangrams.github.io/walkabout-style-more-labels/#18/37.91867/-122.04319>)

(<https://tangrams.github.io/walkabout-style-more-labels/#19/37.74048/-122.43545>)

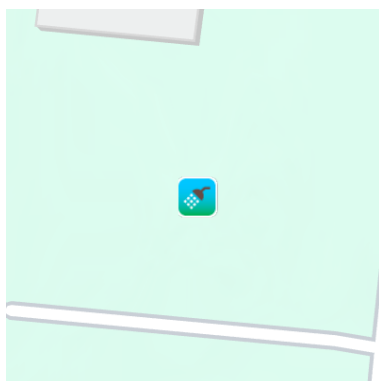
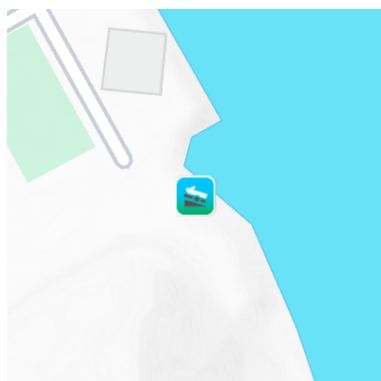
(<https://tangrams.github.io/walkabout-style-more-labels/#20/39.32057/-120.98989>)

(<https://tangrams.github.io/walkabout-style-more-labels/#16/38.8404/-79.3692>)

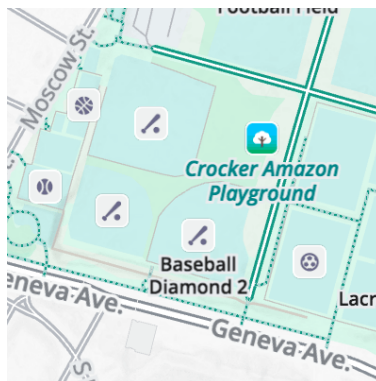
(<https://tangrams.github.io/walkabout-style-more-labels/#17/37.75701/-119.59780>)



(<https://tangram.ms.github.io/walkabout-style-more-labels/#18/37.88966/-122.44594>)



[ms.github.io/walkabout-style-more-labels/#20/37.83244/-122.53933](https://tangram.ms.github.io/walkabout-style-more-labels/#20/37.83244/-122.53933))

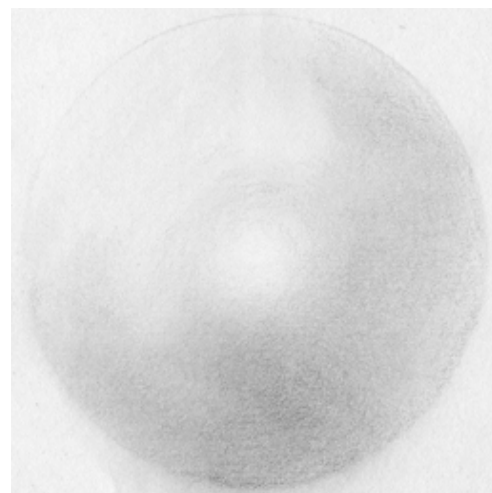


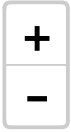
(<https://tangram.ms.github.io/walkabout-style-more-labels/#16/37.7140/-122.4297>)

Hill shading

The pencil drawing at right is the sphere map image that powers hill shading in **Walkabout**.

The map below shows how Tangram uses the sphere map image and elevation normals to generate hill shading for Walkabout. Water is drawn with a solid bright blue to contrast with the grayscale terrain to provide the base upon which all other map layers sit. This feature is powered by Mapzen's global **elevation tiles** (<https://www.mapzen.com/blog/elevation/>).



[Leaflet](#)

Walkabout San Francisco. View full screen ↗ (<https://tangrams.github.io/spheremap-demos/?noscroll&url=styles/walkabout.yaml#10/37.8992/-122.4248>)

Then we colorize the hill shading for green, brown and other park & landuse areas:

[Leaflet](#)

Walkabout San Francisco. View full screen ↗ (<https://tangrams.github.io/spheremap-demos/?noscroll&url=styles/walkabout-landuse.yaml#10/37.8992/-122.4248>)

Adding trails and roads, with early trail visibility powered by **v0.10 vector tile changes** (<https://www.mapzen.com/blog/v0.10-tiles/>):

[Leaflet](#)

Walkabout San Francisco. View full screen ↗ (<https://tangrams.github.io/walkabout-style-no-labels/#12/37.7753/-122.4213>)

All together now, around the world

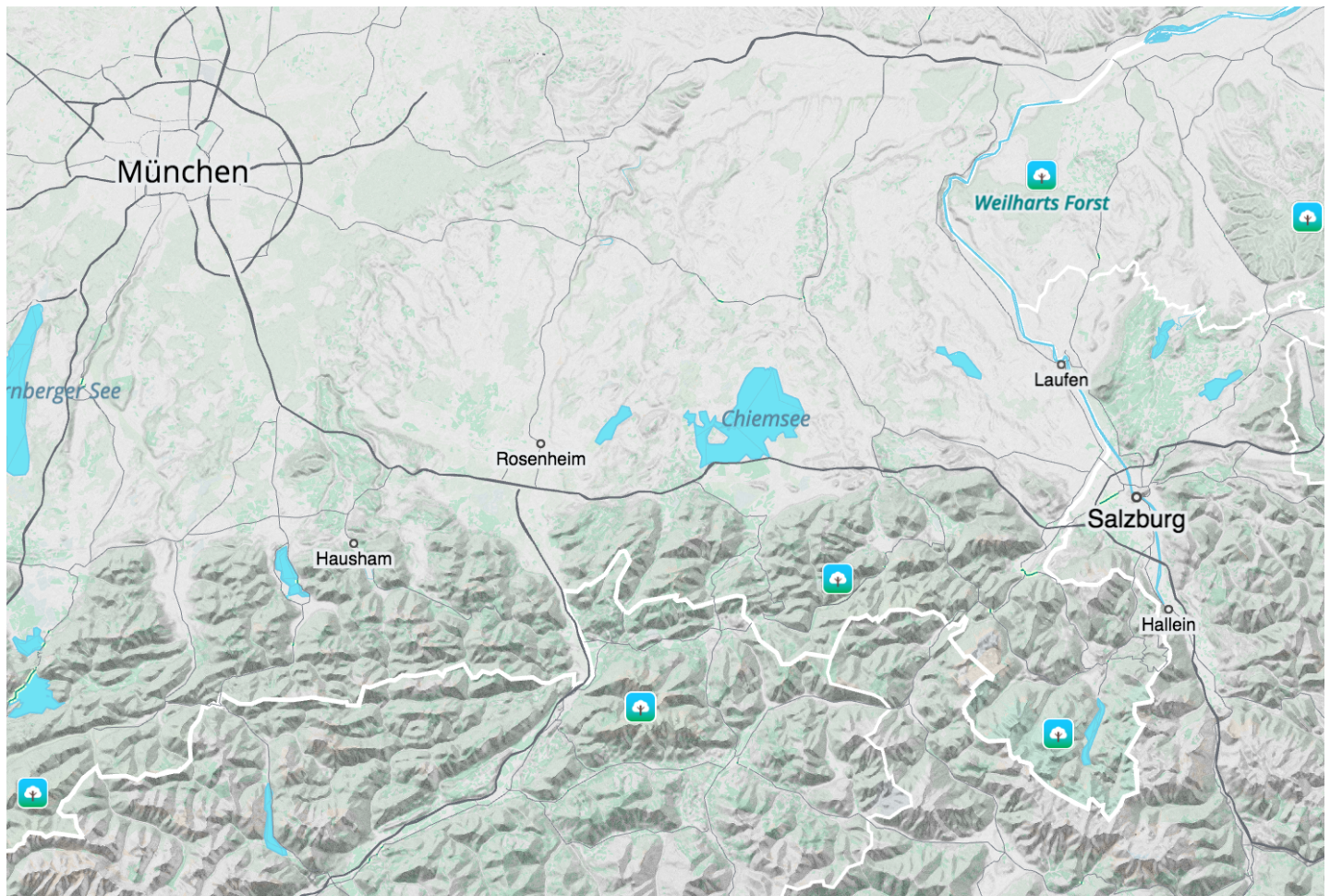
All images below click thru to full screen interactive maps.

Zoom 6: European Alps over to the Carpathian Mountains, and the Appennini in Italy.



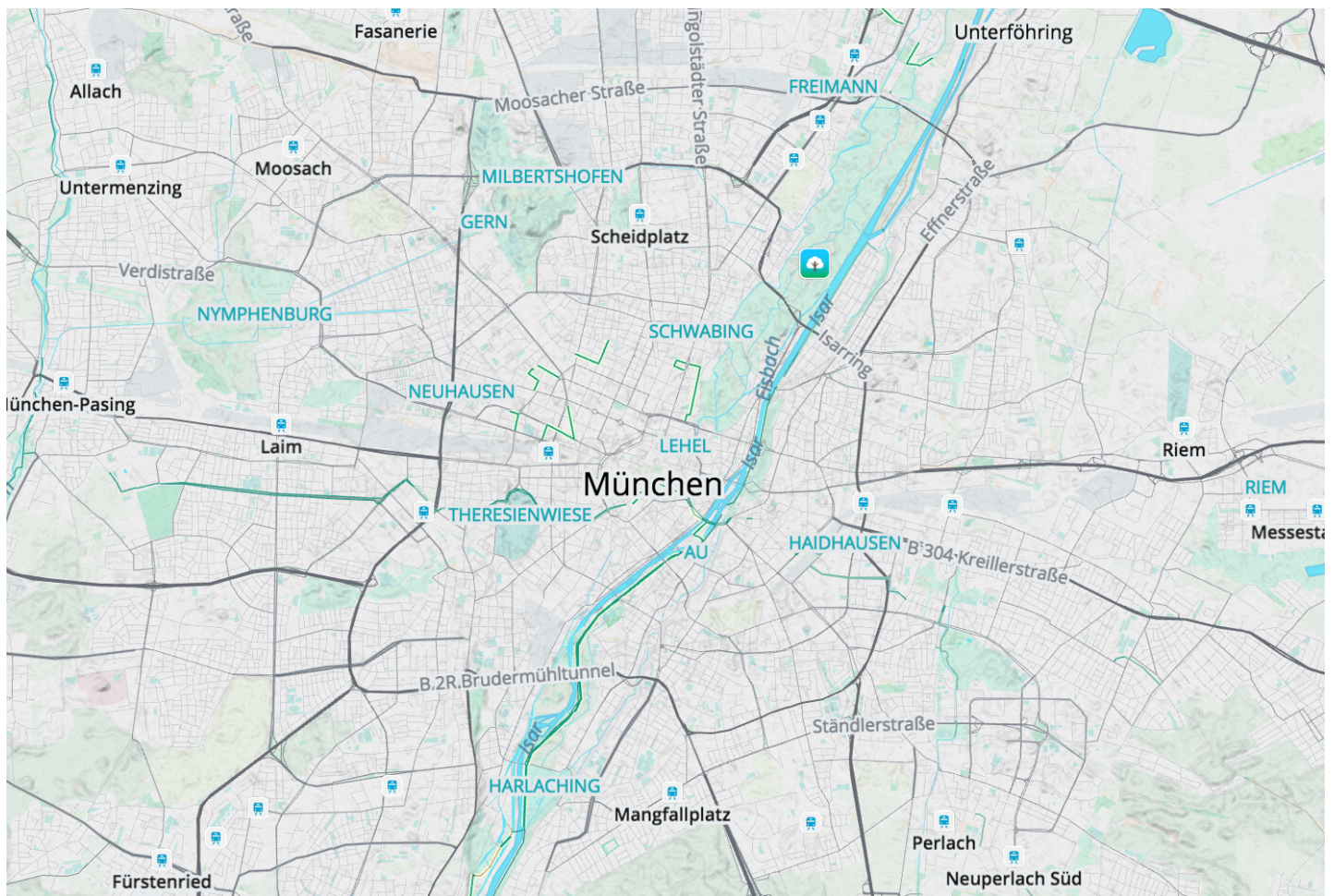
(<https://tangrams.github.io/walkabout-style-more-labels/#6/45.422/-347.388>)

Zoom 9: Munich, Germany and Salzburg, Austria mark the Alp's northern extent.



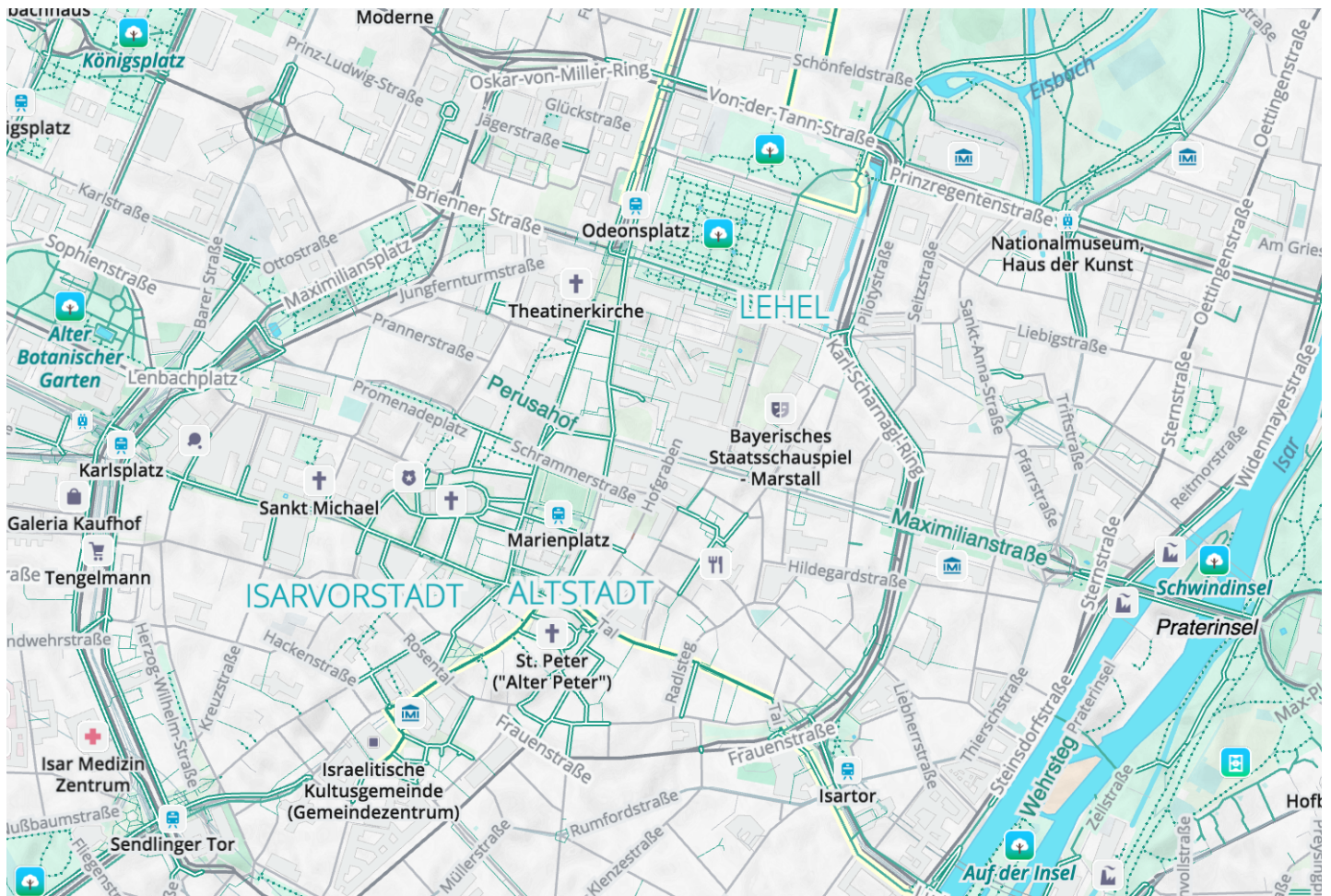
(<https://tangrams.github.io/walkabout-style-more-labels/#9/47.8795/-347.6074>)

Zoom 12: Outdoor recreation options abound in Munich with the English Garden on the western banks of the Isar. Major landmarks like train stations are called out with blue icons and a similar blue is used for neighbourhood labels.



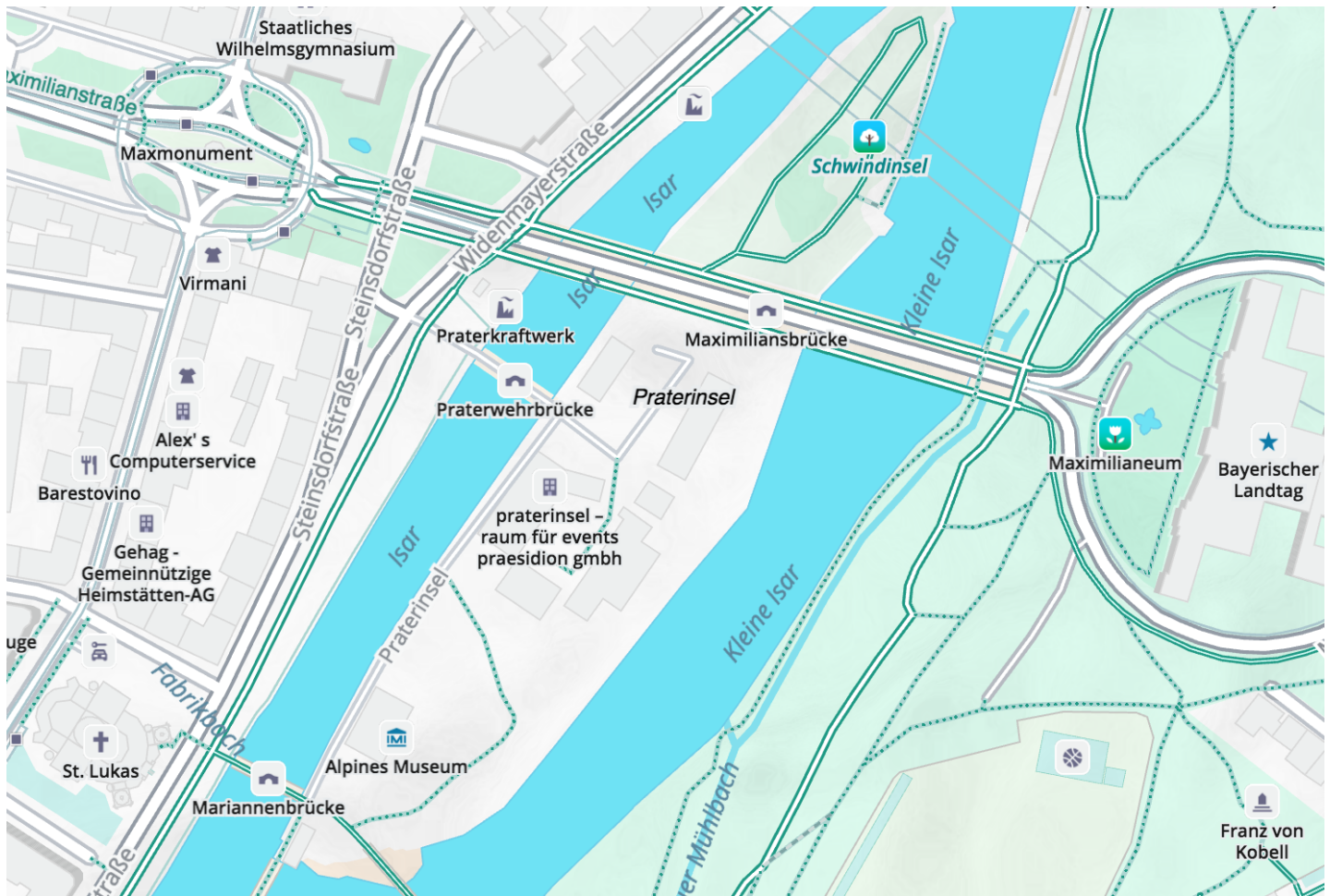
(<https://tangrams.github.io/walkabout-style-more-labels/#12/48.1519/-348.4249>)

Zoom 15: Downtown stretching from the central train station through the old city and onto the Isar river. Shopping galleries, parks, cathedrals, beerhalls, and other major attractions are featured. Museums are blue and hospitals are red. Note the extensive walking network and pedestrian-only zone in the old city!



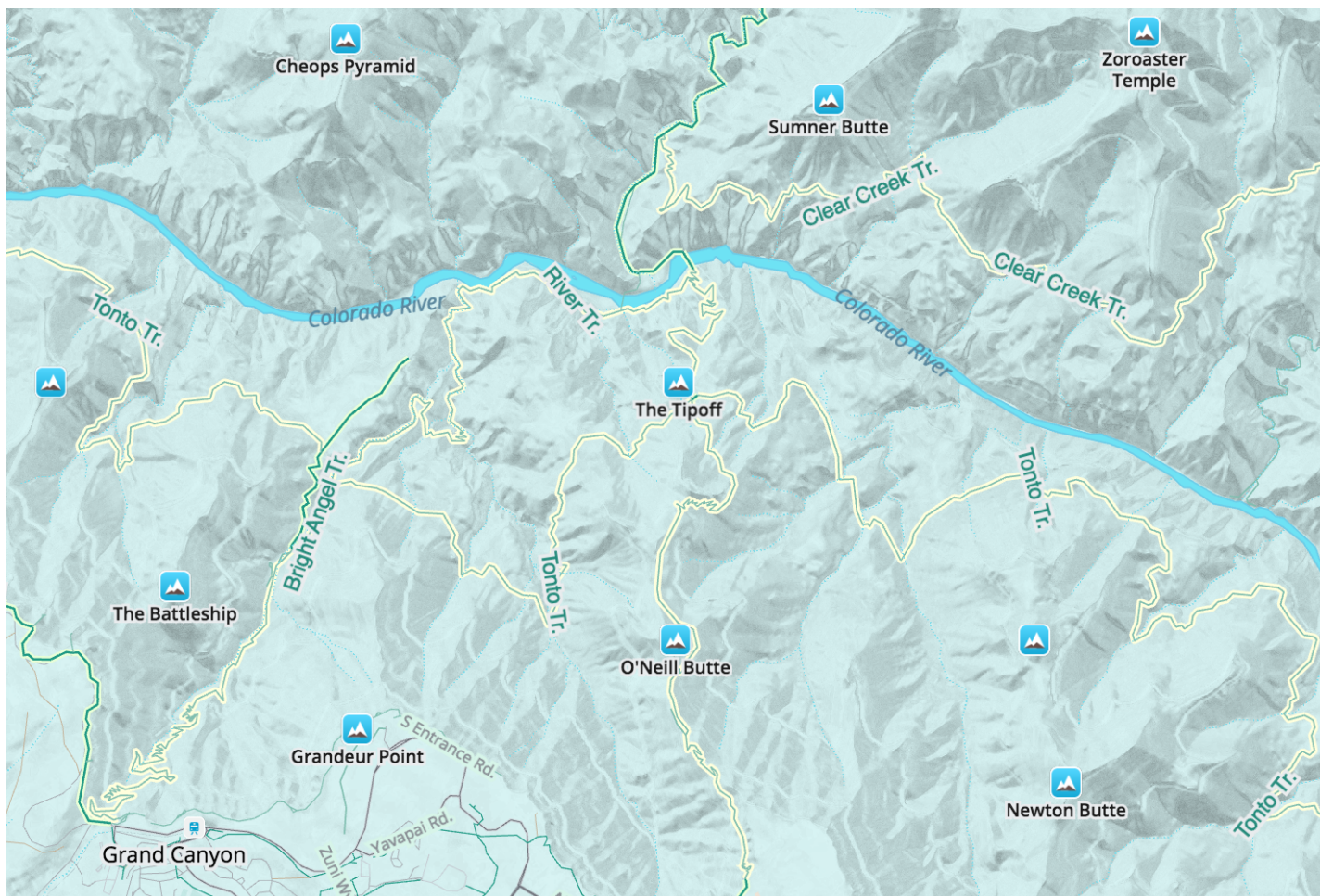
(<https://tangrams.github.io/walkabout-style-more-labels/#15/48.1431/-348.4277>)

Zoom 17: All shops, restaurants, and other attractions are now visible with a muted grey icon treatment. The Alpines Museum on the Praterinsel Island is called out as urban walking attraction.



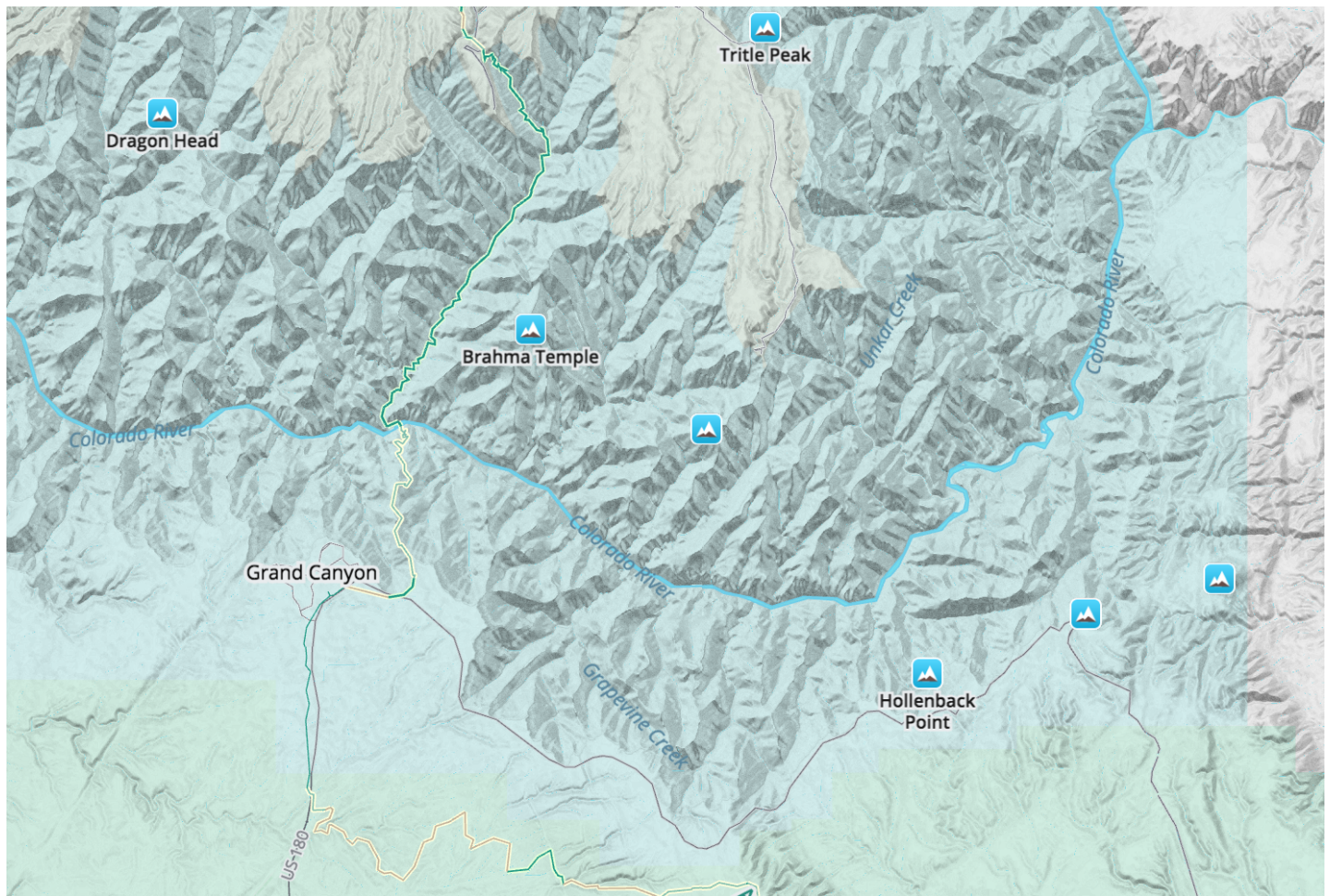
(<https://tangrams.github.io/walkabout-style-more-labels/#17/48.13617/-348.40892>)

Zoom 13: Switching continents, we're now at the Grand Canyon with the Bright Angel Trail leading from the village down, down, down to the Colorado River with Cheops Pyramid and Zoroaster Temple north of the river.



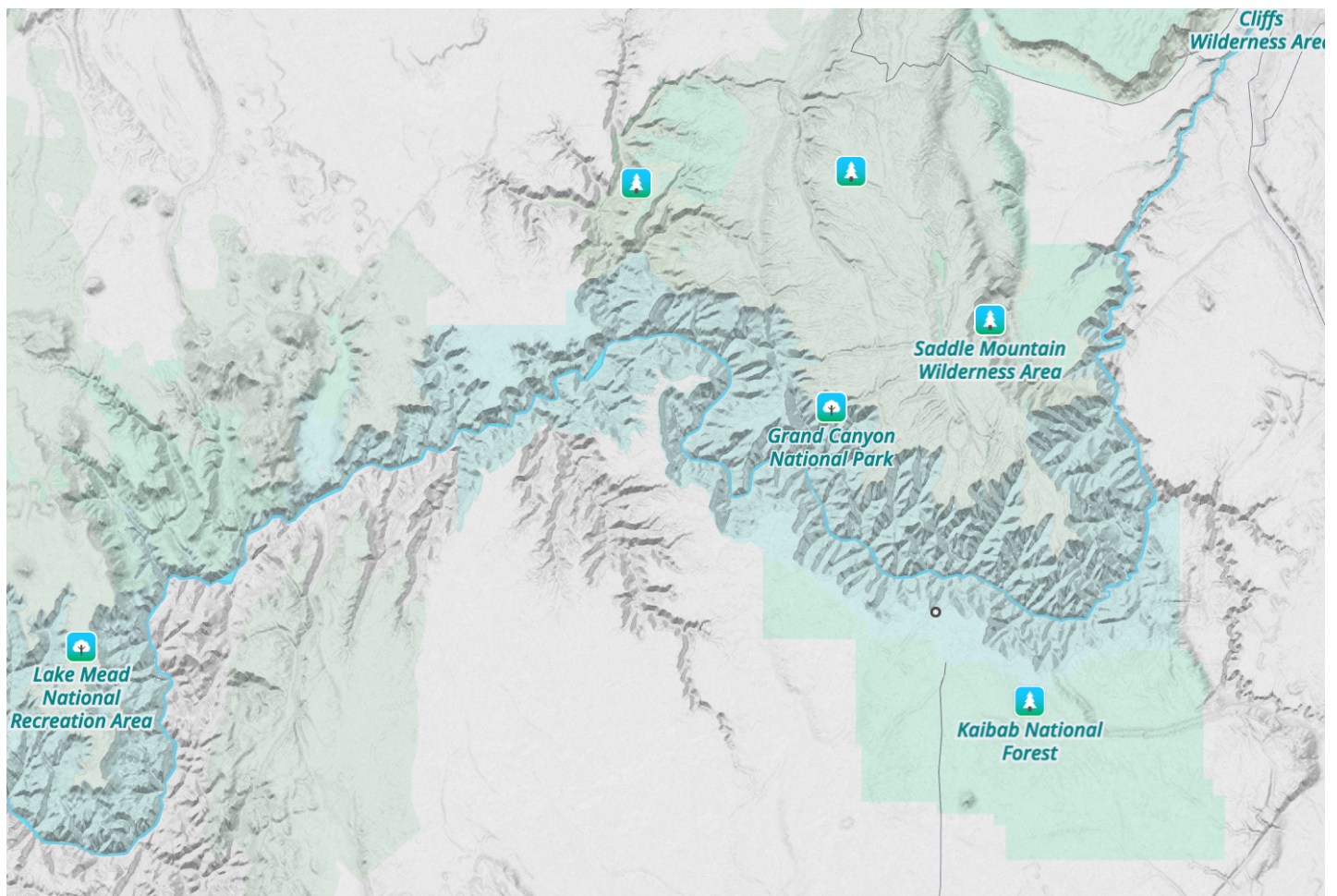
(<https://tangrams.github.io/walkabout-style-more-labels/#13/36.0817/-112.1179>)

Zoom 11: Stepping back, we see the full extent of the canyon.



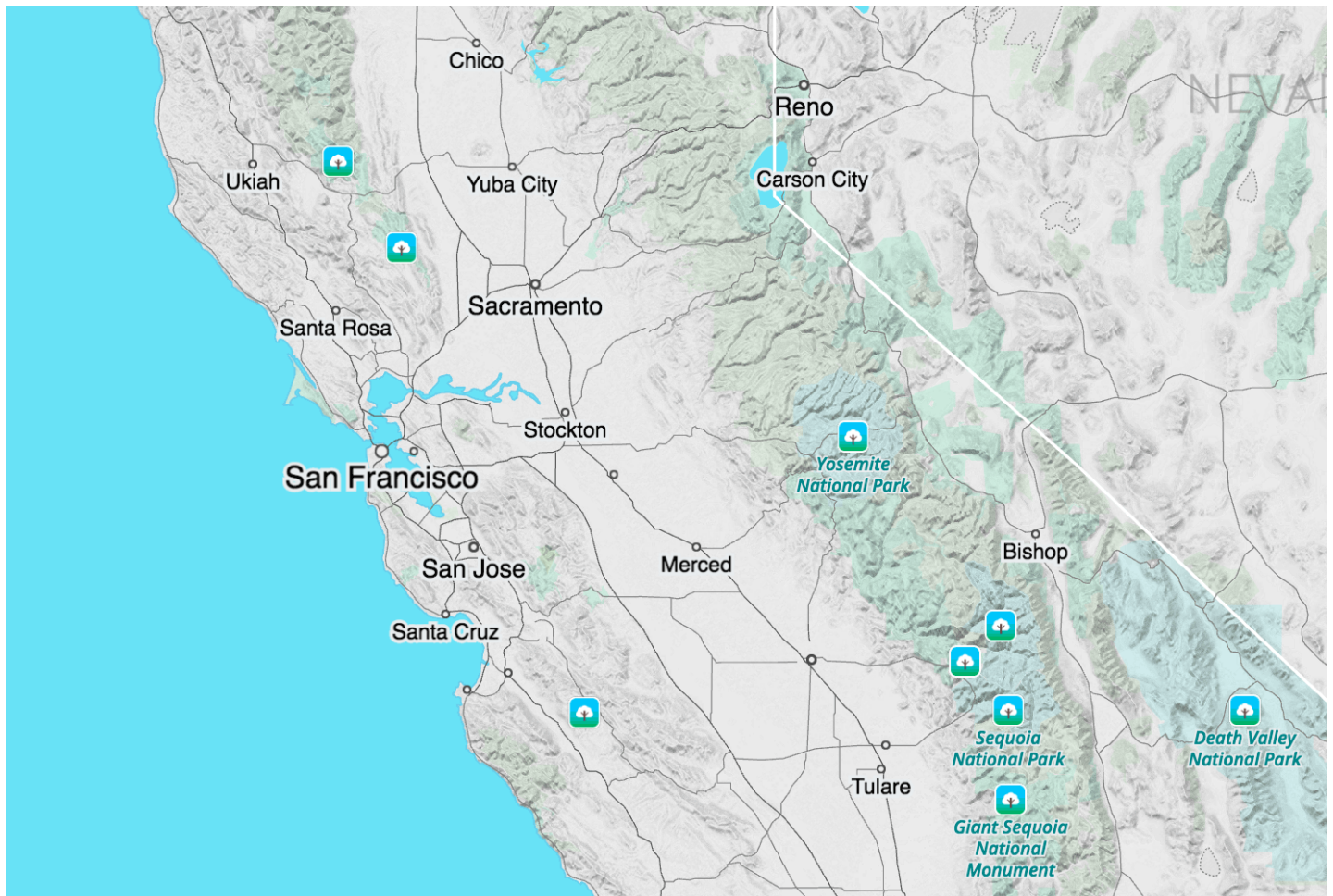
(<https://tangrams.github.io/walkabout-style-more-labels/#11/36.1495/-112.0825>)

Zoom 9: The mighty Colorado bracketed by Lake Powell upstream and Lake Mead downstream.



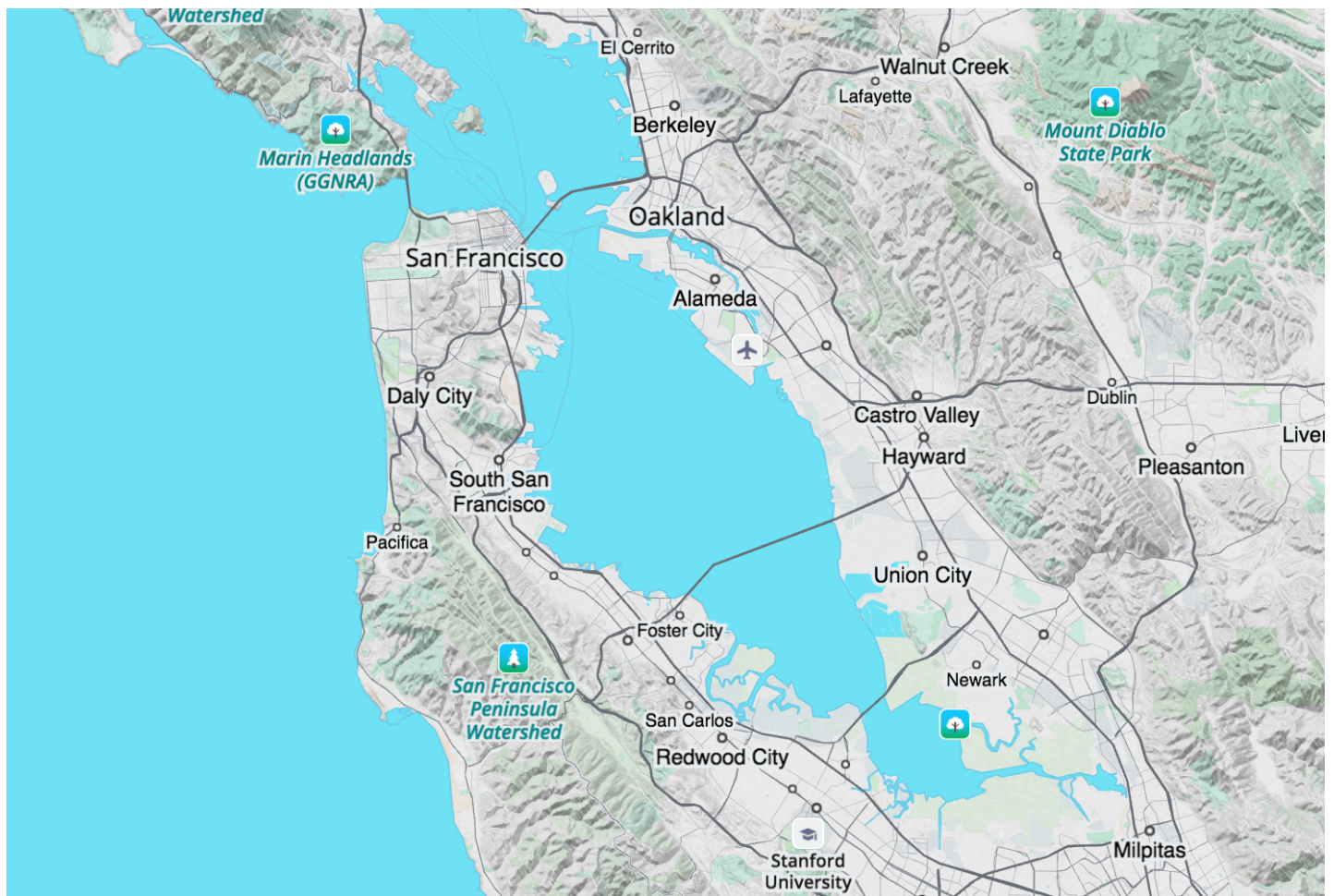
(<https://tangrams.github.io/walkabout-style-more-labels/#9/36.4655/-113.1729>)

Zoom 7: Panning over to California we see the snow crested Sierra Nevada, the Central Valley, and San Francisco sitting on the Pacific Ocean.



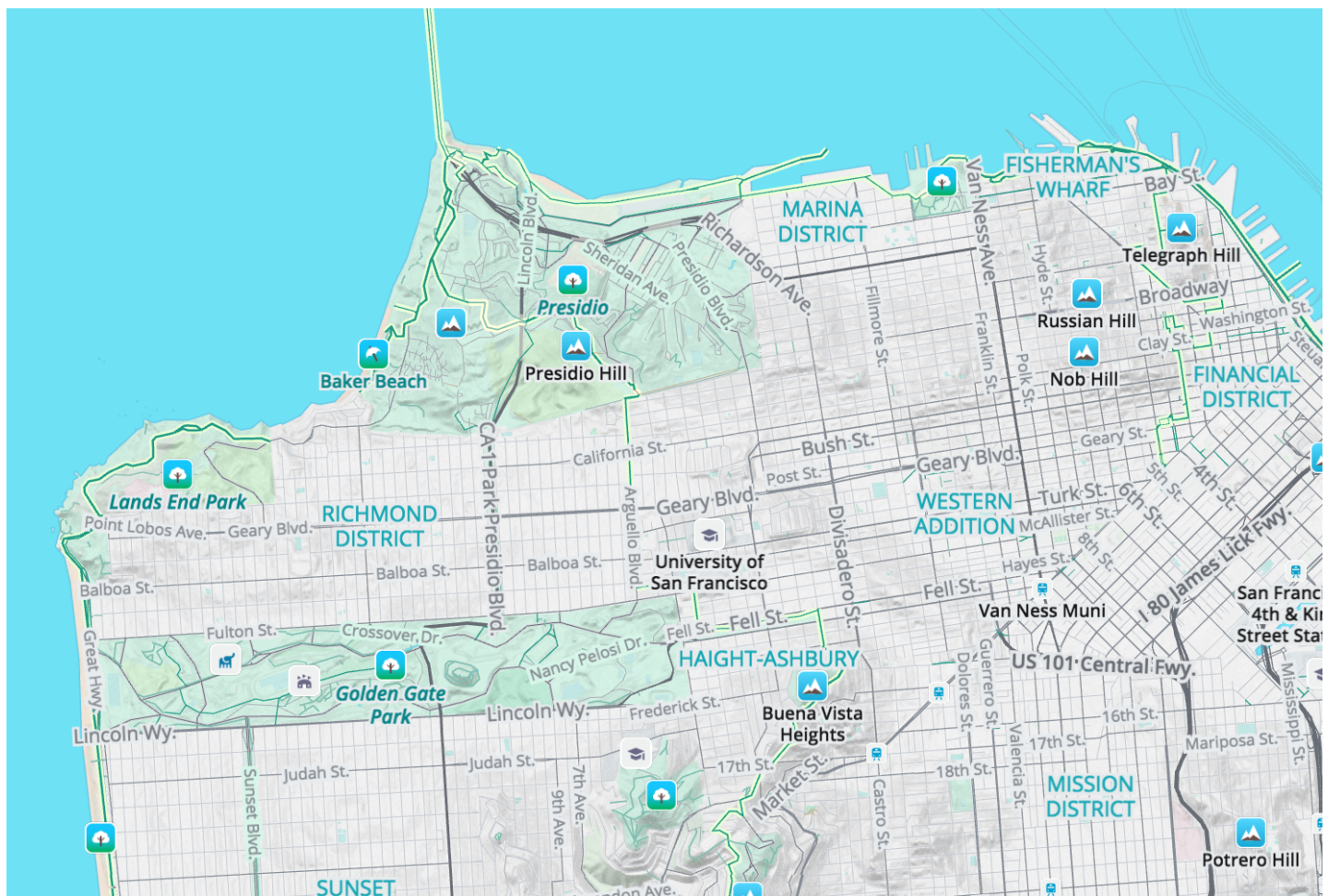
(<https://tangrams.github.io/walkabout-style-more-labels/#7/38.255/-120.454>)

Zoom 10: Let's spend some time in the San Francisco Bay Area, with Mt. Diablo in the east.



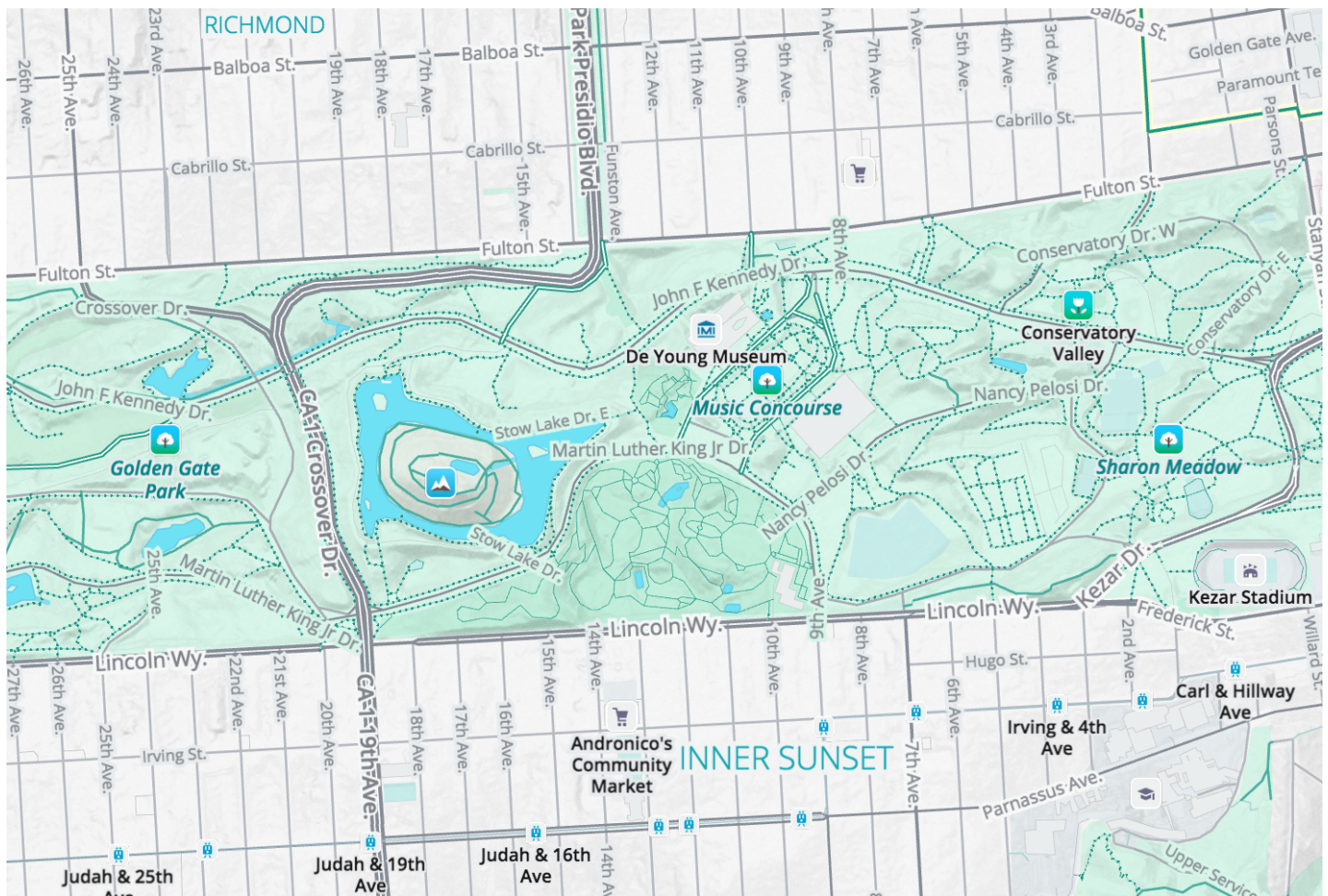
(<https://tangrams.github.io/walkabout-style-more-labels/#10/37.7588/-122.2174>)

Zoom 13: From Bay to Breakers and a few hills in between San Francisco offers the Bay trail, the Barbary Coast trail, Bay Ridge trail, and the California Coastal trail in green with a yellow highlight that in later zooms helps you stay on route thru areas of dense footpaths.



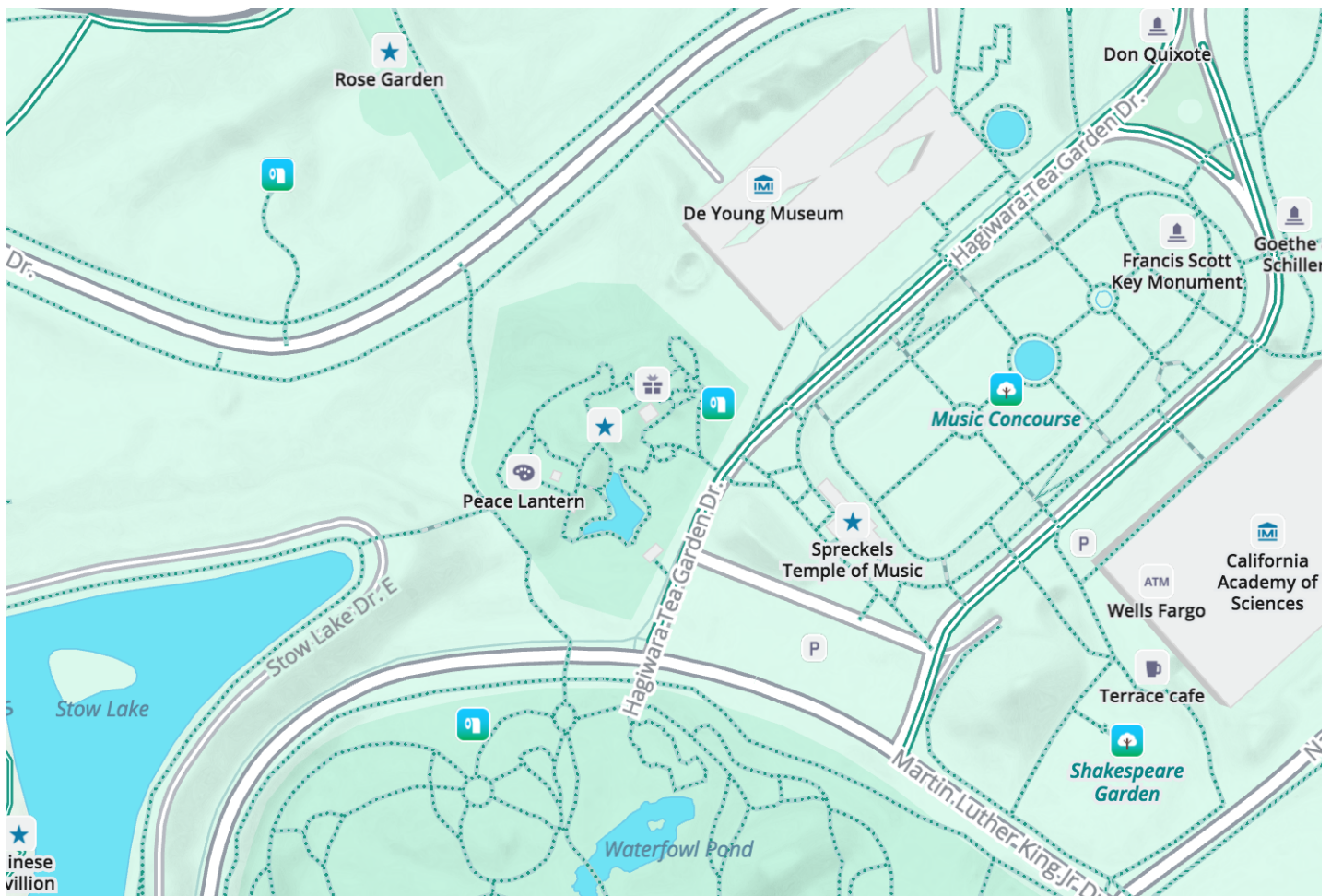
(<https://tangrams.github.io/walkabout-style-more-labels/#13/37.7996/-122.4629>)

Zoom 15: Golden Gate Park was founded in the 1870s and rivals New York's Central Park in total area and attractions with bike paths and foot trails connecting Ocean Beach with the urban city scape and famous museums like the De Young.



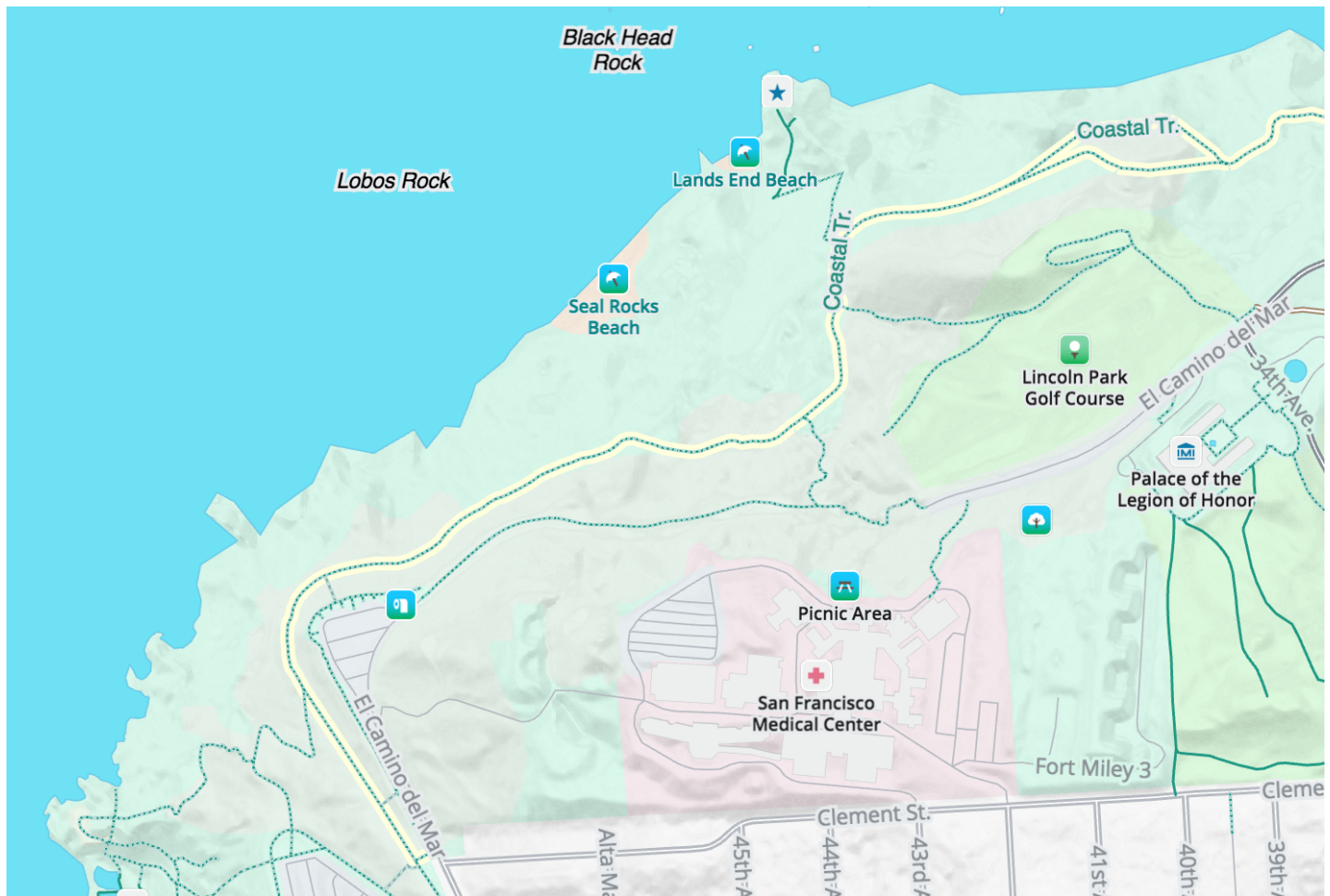
(<https://tangrams.github.io/walkabout-style-more-labels/#15/37.7695/-122.4830>)

Zoom 17: The heart of the park around the De Young Museum, California Academy of Sciences, and Stowe Lake is full of gardens, picnic spots, and restrooms.



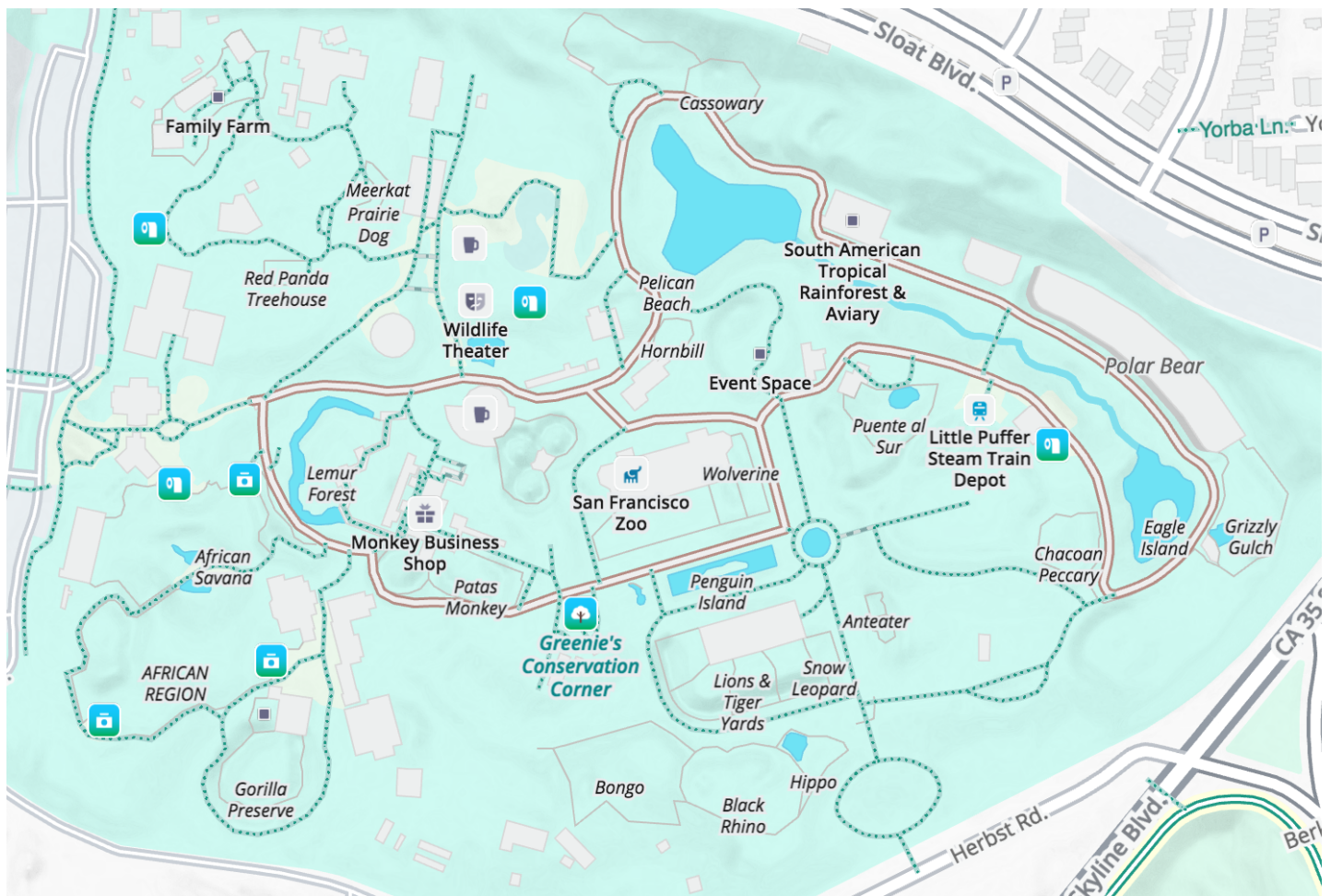
(<https://tangrams.github.io/walkabout-style-more-labels/#17/37.77025/-122.47137>)

Zoom 17: Panning over to Lands End just north of Golden Gate Park in San Francisco we find museums, golf courses, hospitals, picnic spots, scenic vistas, beaches, and more toilets all connected by the California Coastal trail.



(<https://tangrams.github.io/walkabout-style-more-labels/#16/37.7836/-122.5052>)

Zoom 17: At the southern end of Ocean Beach the San Francisco Zoo walks visitors from the Lemur Forest to Grizzly Gulch.



(<https://tangrams.github.io/walkabout-style-more-labels/#17/37.73319/-122.50294>)

Zoom 12: Mark Twain once famously quipped, “the coldest winter I ever spent was my summer in San Francisco, but when the going got rough I headed to North Star.” Here you can see Squaw Valley and other ski resorts with the Pacific Crest and Tahoe Rim trails nearby.



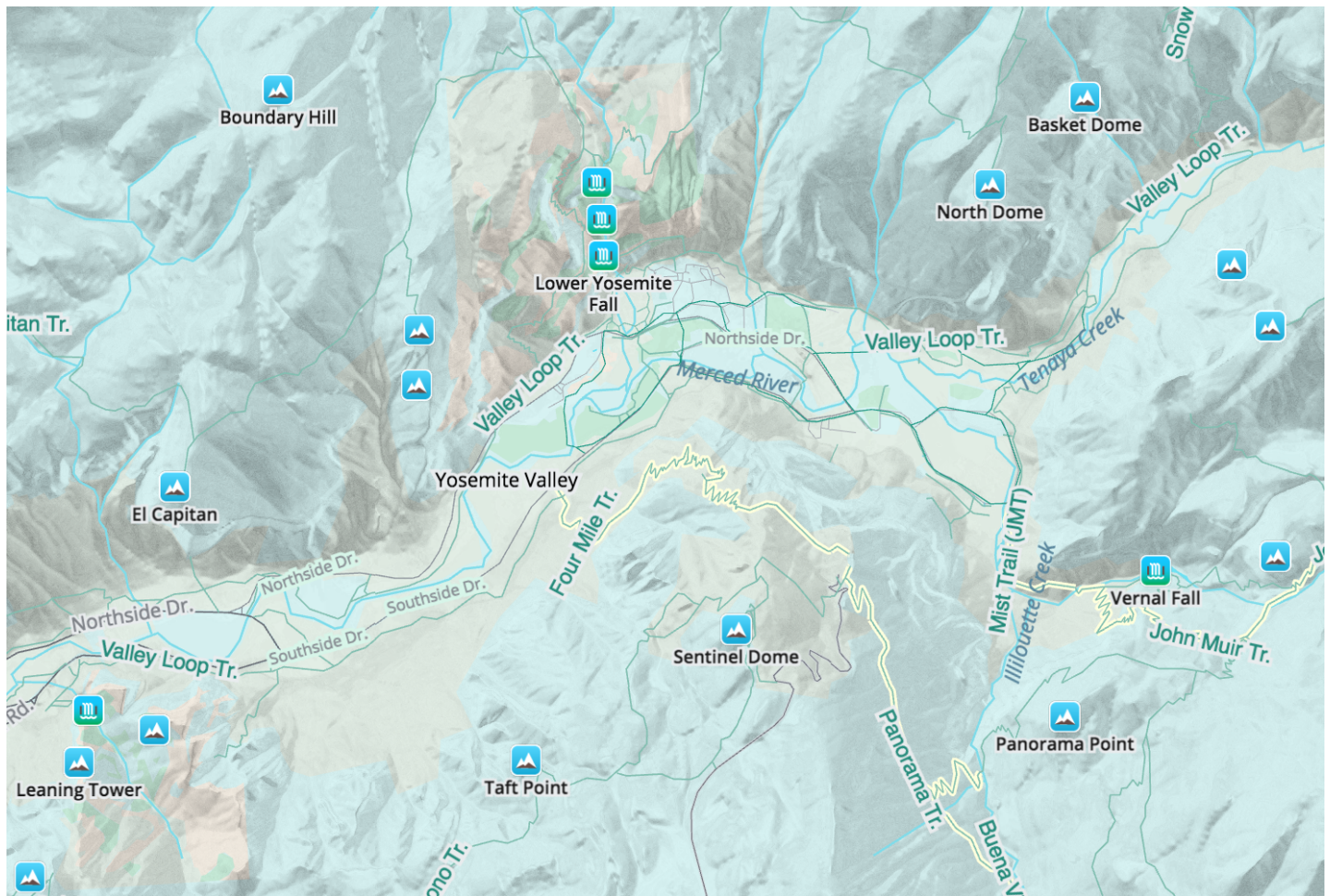
(<https://tangrams.github.io/walkabout-style-more-labels/#12/39.1820/-120.1702>)

Zoom 12: Yosemite is beautiful year round with El Capitan and the falls dominating the valley. On the east side of the valley the John Muir trail wraps behind Half Dome and heads up to the sierra crest.



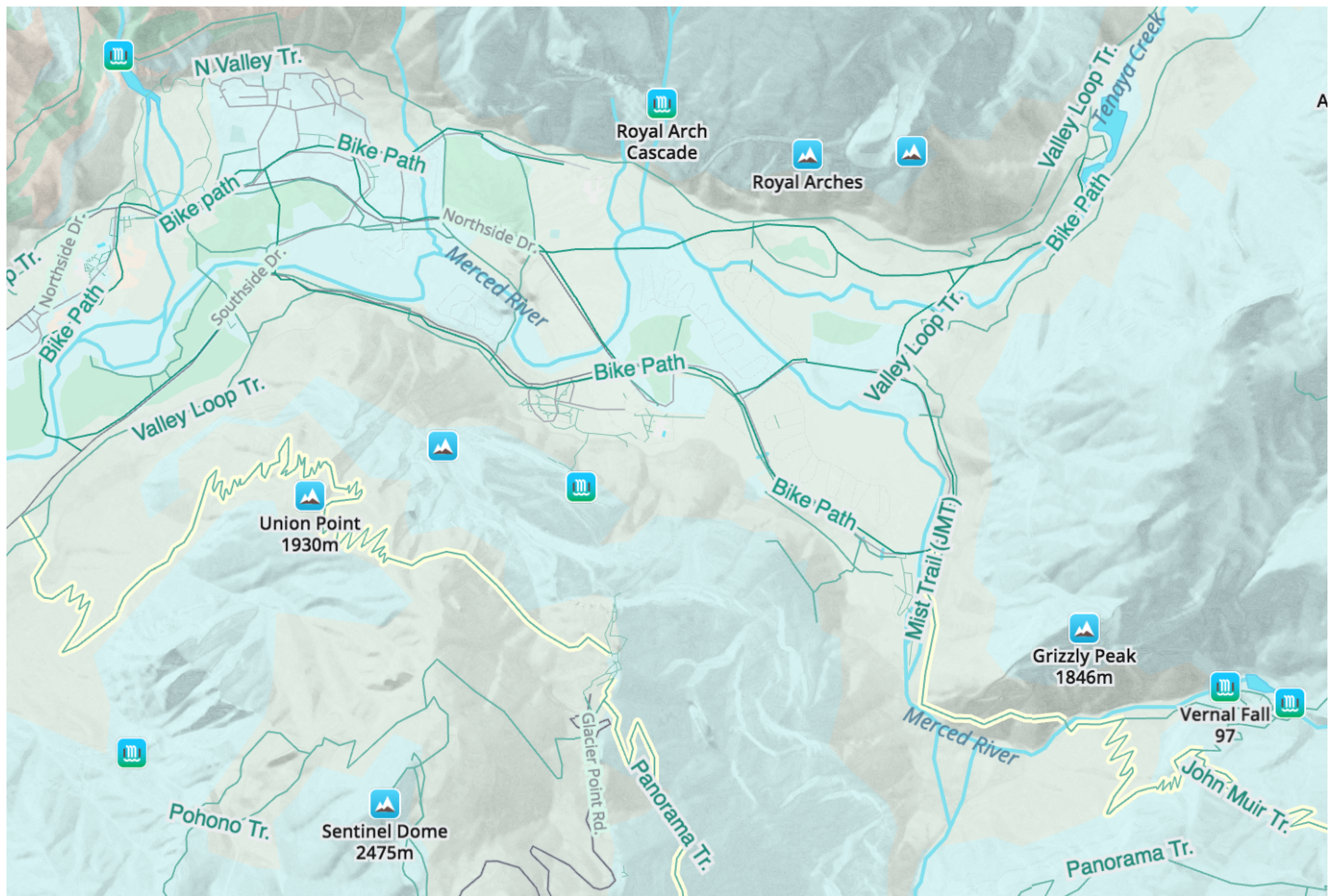
(<https://tangrams.github.io/walkabout-style-more-labels/#12/37.7202/-119.6339>)

Zoom 13: Stepping in for a closer view more trails are shown along with attractions like Bridalveil Falls, Vernal Fall, and Artists Point.



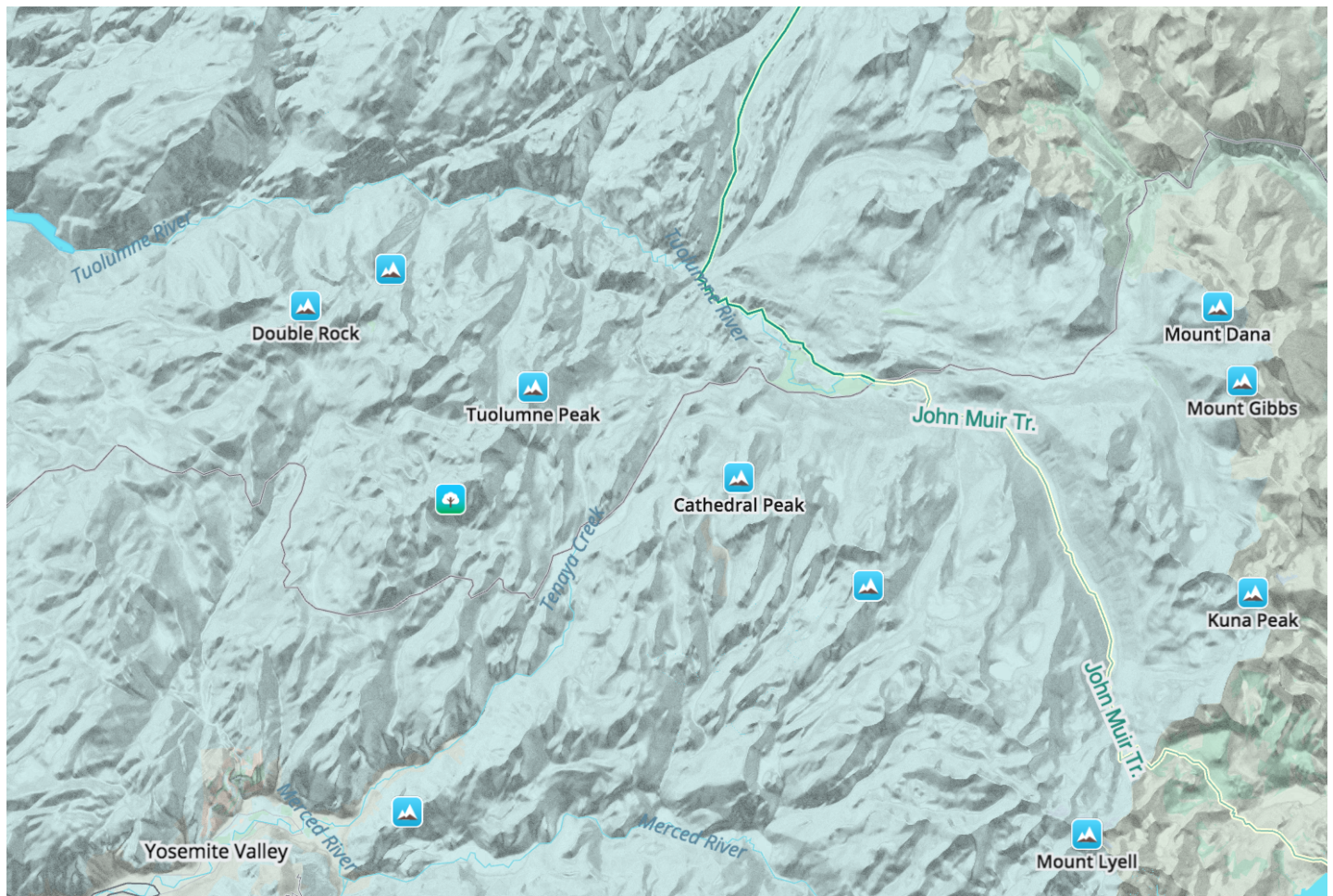
(<https://tangrams.github.io/walkabout-style-more-labels/#13/37.7343/-119.6315>)

Zoom 14: A closer look at Yosemite valley.



(<https://tangrams.github.io/walkabout-style-more-labels/#14/37.7392/-119.5649>)

Zoom 11: The Pacific Crest and John Muir trails are the “interstate” system of the backcountry and Walkabout shows them along side the highways leading to their trailheads.

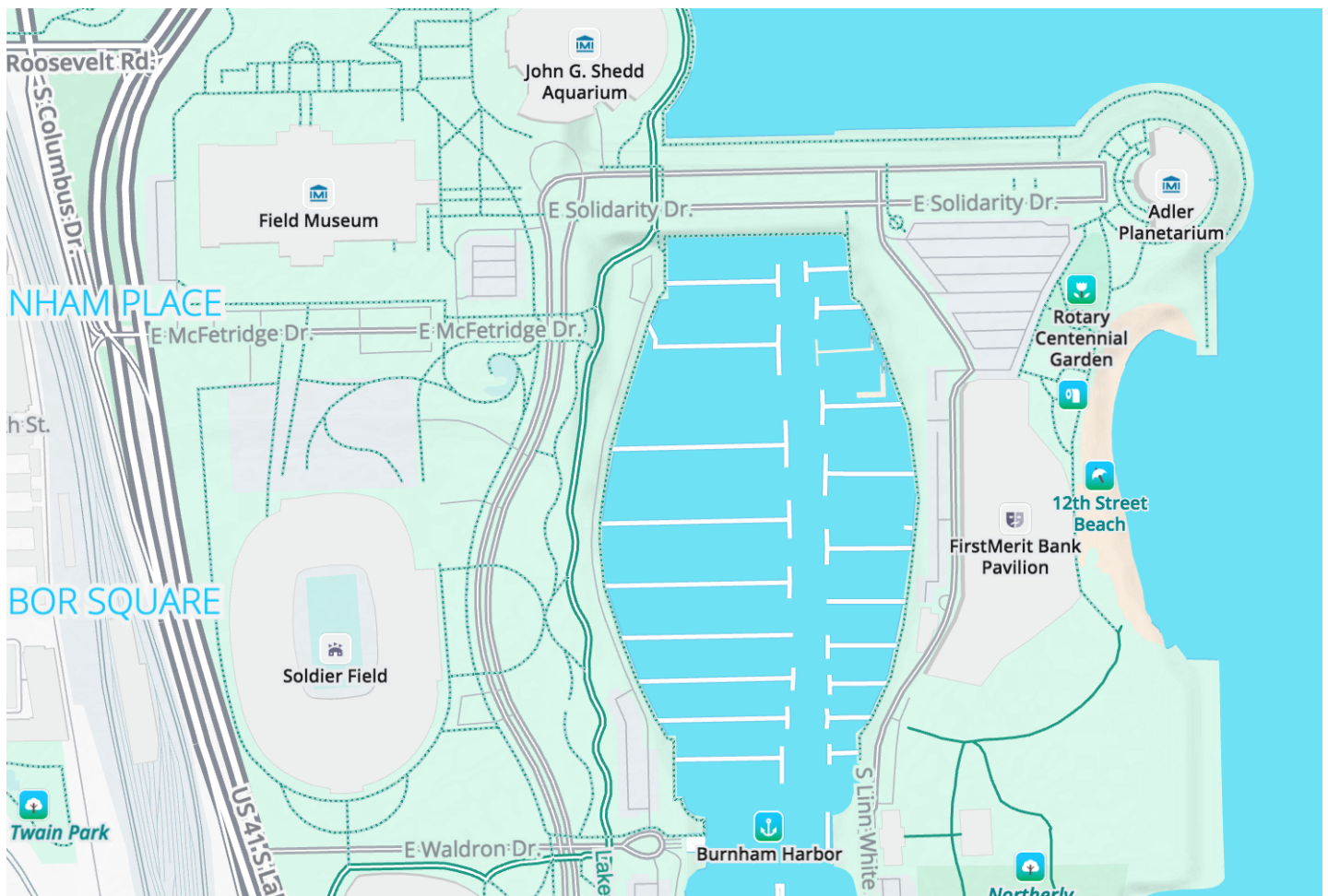


(<https://tangrams.github.io/walkabout-style-more-labels/#11/37.9150/-119.3946>)

Zoom 15: Central Park in New York city.

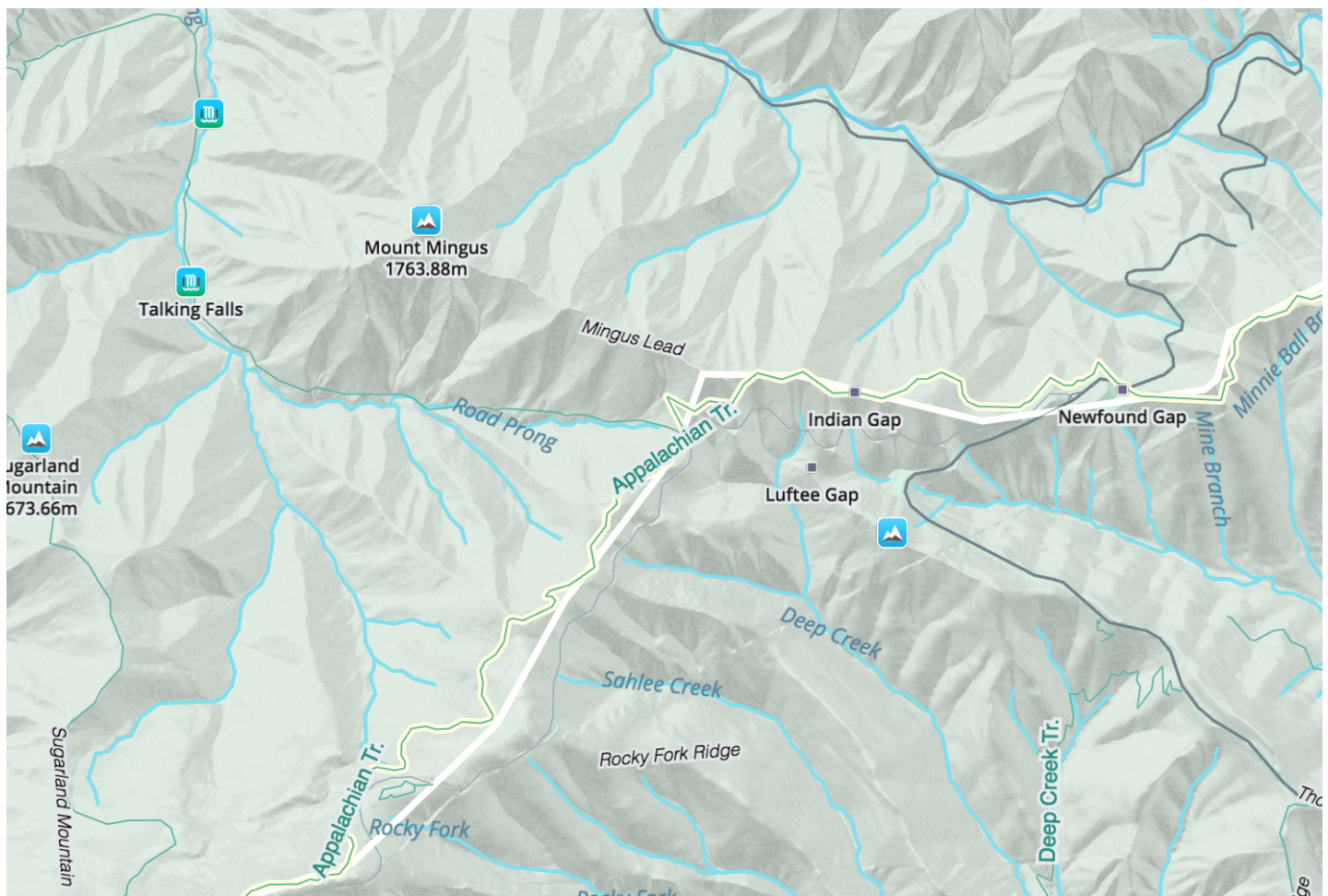


<https://mapzen.com/blog/walkabout/>



(<https://tangrams.github.io/walkabout-style-more-labels/#16/41.8635/-87.6177>)

Zoom 14: Newfound Gap on the Appalachian Trail in Great Smokey Mountains National Park. Ridges and creeks are named.



(<https://tangrams.github.io/walkabout-style-more-labels/#14/35.6090/-83.4596>)

Zoom 11: Honolulu, Hawaii.



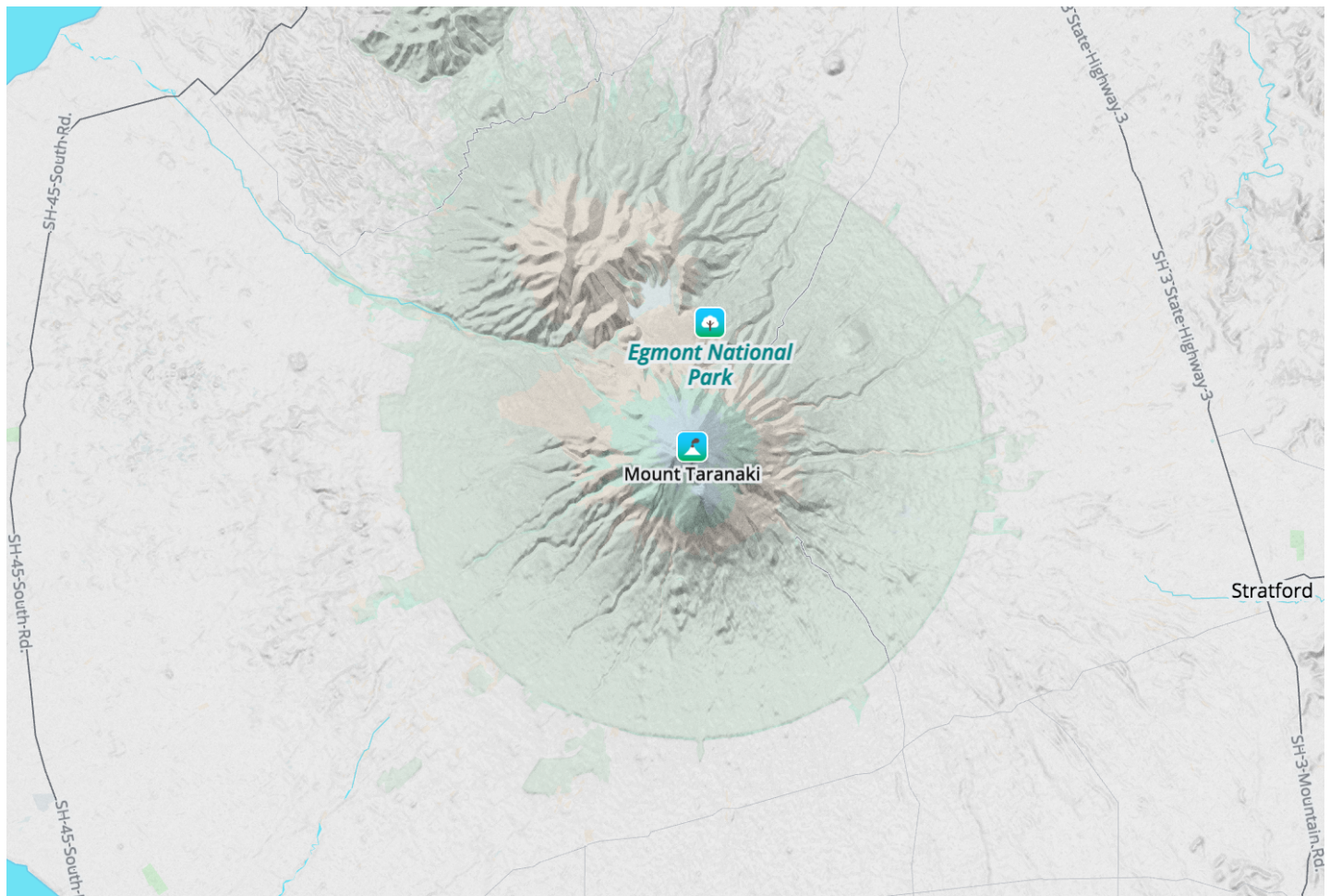
(<https://tangrams.github.io/walkabout-style-more-labels/#11/21.4940/-157.9910>)

Zoom 12: Table Mountain and Cape Town, South Africa.



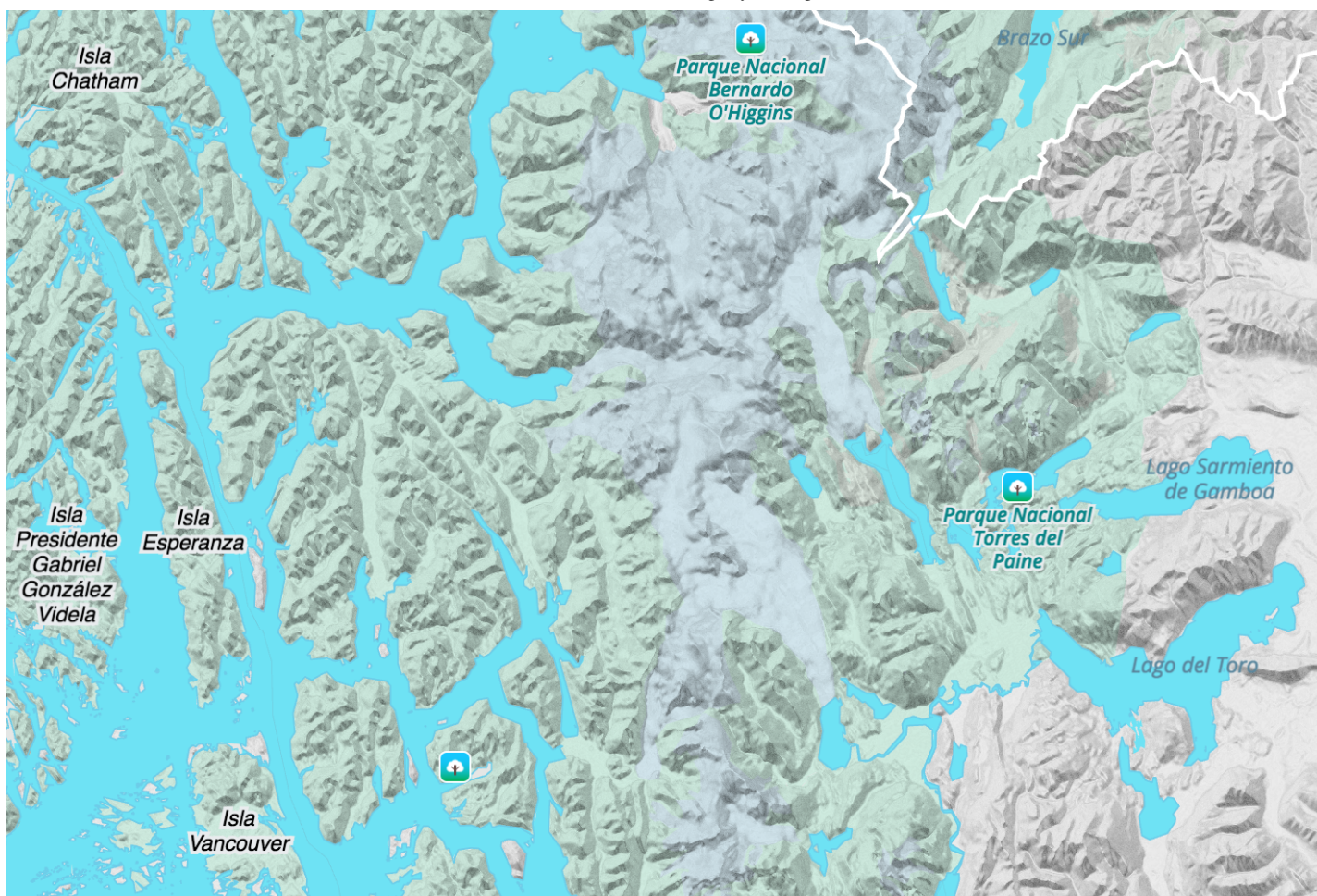
(<https://tangrams.github.io/walkabout-style-more-labels/#12/-33.9633/18.4320>)

Zoom 11: Mt. Taranaki in New Zealand's Egmont National Park.



(<https://tangrams.github.io/walkabout-style-more-labels/#11/-39.2679/173.9960>)

Zoom 9: Torres del Paine National Park in southern Chile.



(<https://tangrams.github.io/walkabout-style-more-labels/#9/-51.0569/-73.6606>)

Zoom 11: Mount Fuji in Japan.



(<https://tangrams.github.io/walkabout-style-more-labels/#11/35.3196/138.7312>)

Zoom 10: Taal volcano south of Manila, Philippines.



(<https://tangrams.github.io/walkabout-style-more-labels/#10/14.2005/121.1147>)

Zoom 11: Mount Everest surrounded by glaciers.



(<https://tangrams.github.io/walkabout-style-more-labels/#11/28.0053/86.9472>)

But enough of the walk thru. Please take a **Walkabout** and *get outside!*

· 01 July 2016 ·



Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.



Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

© 2017 Mapzen

Sphere Maps

tangram (</tag/tangram>) **demo** (</tag/demo>) **data** (</tag/data>)

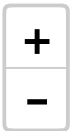
cartography (</tag/cartography>)

In creating our first outdoor style, I discovered the wonderful world of sphere maps! If you haven't read Peter's blog post **Mapping Mountains** (<https://mapzen.com/blog/mapping-mountains/>), please do so. It provides a historical overview on the mapping of mountains and also goes over Mapzen's elevation data in detail and the various techniques we can use to display elevation on maps.

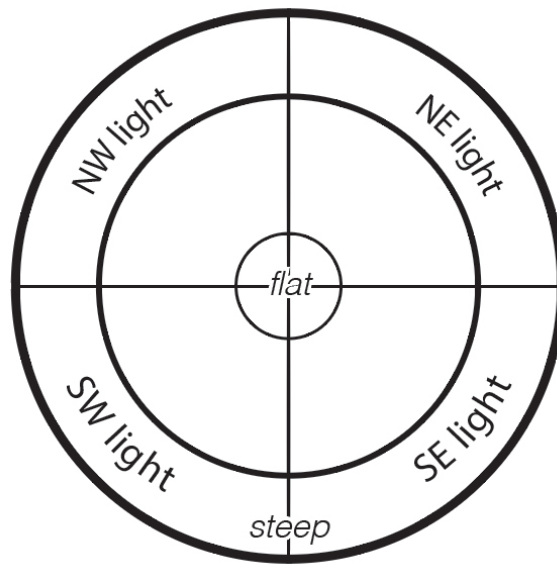
After exploring the various options Peter outlined on his blog post, I chose to use the sphere map technique for our first outdoor style called **Walkabout** (<https://tangrams.github.io/walkabout-style-more-labels/#12/37.8773/-121.9290>). The technique of sphere mapping is a familiar one in video games and film VFX. It's an inexpensive way to add detail to 3D objects by using a texture image, but instead here we're using the texture to create the lighting on the map. Imagine, a 256x256 pixel image can compose the lighting for an entire map! The process of translating pixels from micro to macro is completely fascinating.

Let's go straight to my experiments below and discover what sphere maps are all about. In my curiosity, I tested out various mediums from pencils to watercolors to see what kind of textures would emerge on the map. Each iteration was a delightful surprise. Sometimes I thought I could predict the outcome and would work towards that picture in my mind. Other experiments were pure tests wherein I had no expectation whatsoever about a certain result.

Here's one of the first sketches I did with pencil.

[Leaflet](#)

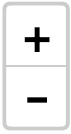
Pencil (These maps are interactive! Open full screen ↗ (<http://tangrams.github.io/spheremap-demos?url=styles/pencil2.yaml#12/37.8773/-121.9290>))



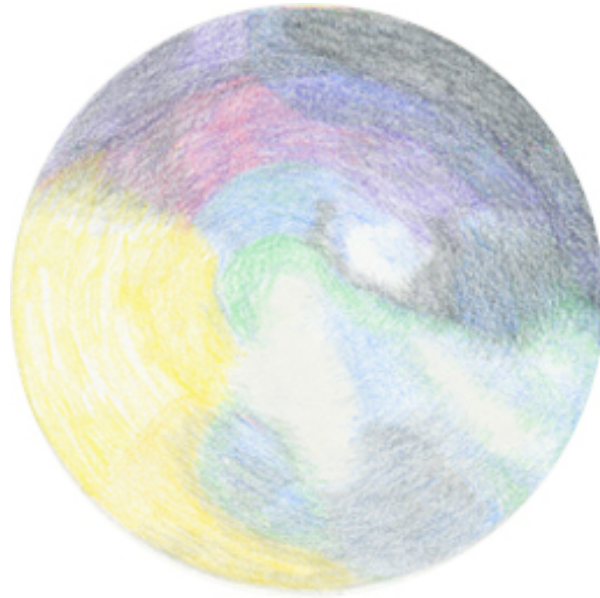
Light chart by Nathaniel Vaughn Kelso

Nathaniel's light chart above could help you picture how the light is being drawn onto the map. I like how the paper grain from the first pencil sketch came through on the map. For the next test, I tried a smudged pencil drawing.

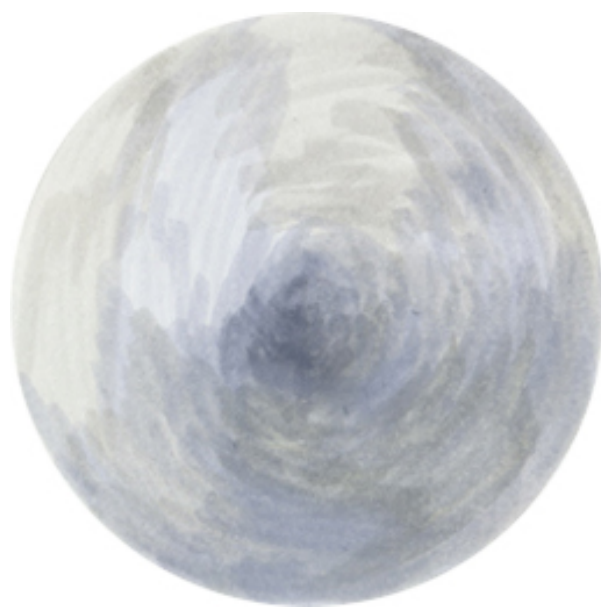


[Leaflet](#)

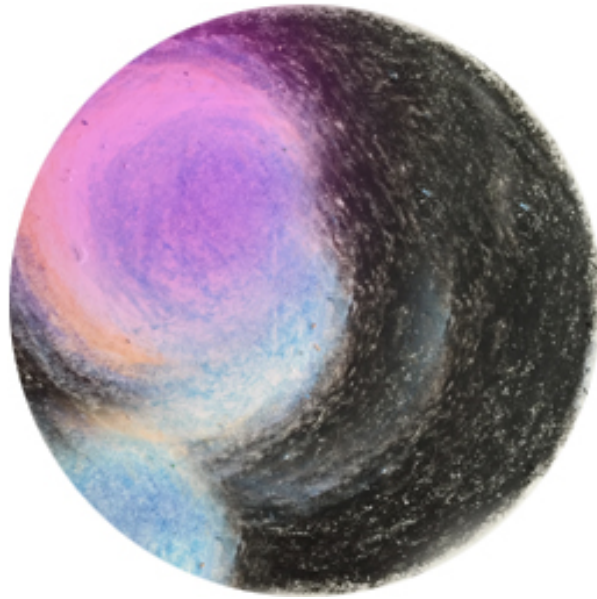
Smudged pencil (These maps are interactive! Open full screen ↗ (<http://tangrams.github.io/spheremap-demos/?url=styles/pencil.yaml#12/37.8773/-121.9290>))



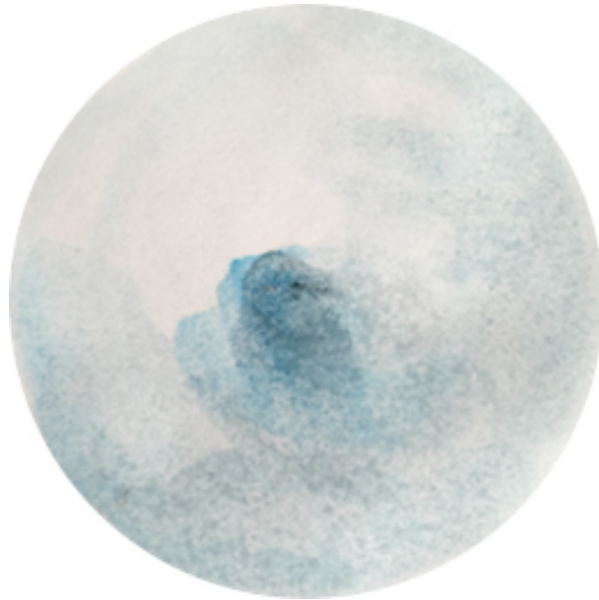
Colored pencils (View the map Open full screen ↗ (<http://tangrams.github.io/spheremap-demos/?url=styles/color-pencil.yaml#12/37.8773/-121.9290>))



Markers (View the map Open full screen ↗ (<http://tangrams.github.io/spheremap-demos/?url=styles/marker.yaml#12/37.8773/-121.9290>))

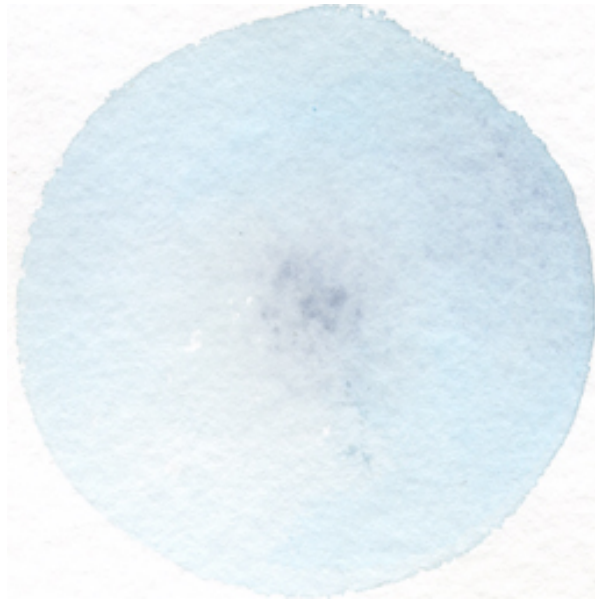


Crayons (View the map Open full screen ↗ (<http://tangrams.github.io/spheremap-demos/?url=styles/crayon.yaml#12/37.8773/-121.9290>))



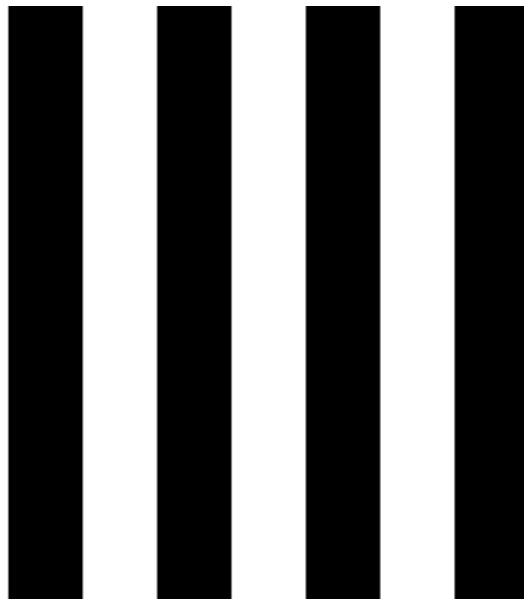
Watercolor (*View the map* *Open full screen* ↗ (<http://tangrams.github.io/spheremap-demos/?url=styles/watercolor2.yaml#12/37.8773/-121.9290>))

With watercolors, you can create a natural smooth gradient, which is perfect for sphere maps.



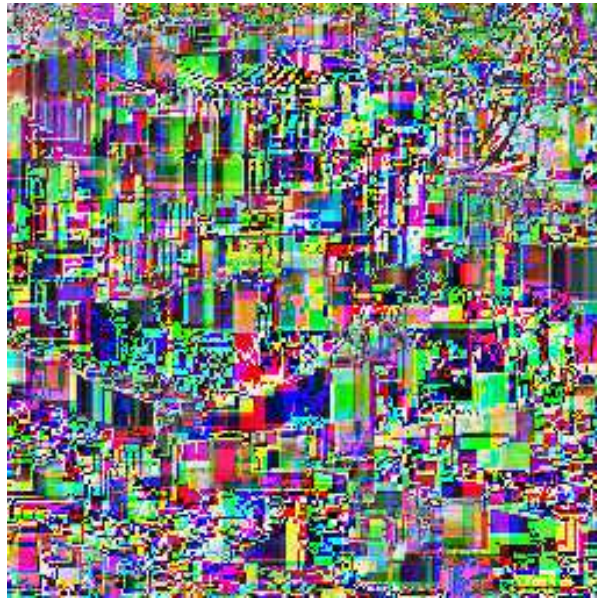
Watercolor (View the map Open full screen ↗ (<http://tangrams.github.io/spheremap-demos?url=styles/watercolor.yaml#12/37.8773/-121.9290>))

I had to try stripes! Surprisingly they work!



Stripes (*View the map* *Open full screen* ↗ (<http://tangrams.github.io/spheremap-demos?url=styles/stripes.yaml#12/37.8773/-121.9290>))

Glitch produces a marbling effect at higher zooms.



Glitch (View the map Open full screen ↗ (<https://tangrams.github.io/spheremap-demos?url=styles/glitch.yaml#12/37.8773/-121.9290>*))*

Circles

Another intriguing feature about sphere maps is that it comes in the form of a flat circle - transforming into a sphere then translating to the lighting on the map. The circle is a universal symbol. It has multiple levels of meaning in various cultures and mythologies. We find circles all over nature. Images of our sun, our planet and planetary movements come to mind. The cycle of time, the growth pattern of trees, the shape of ripples when you throw a rock on the water. What about the shape of bubbles?

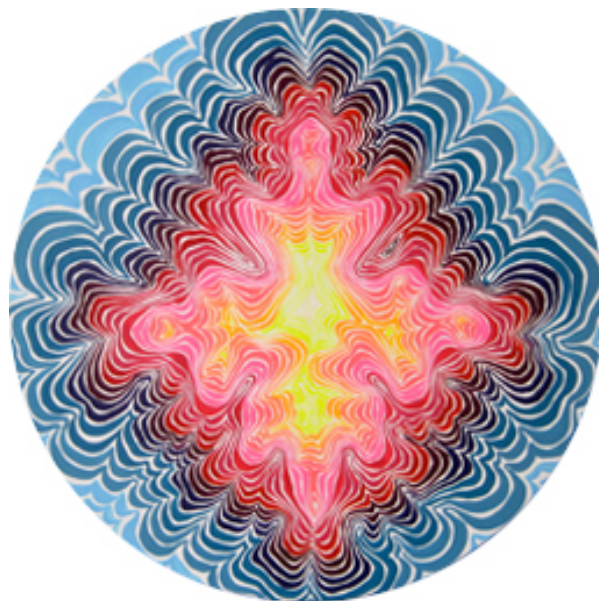
This particular sphere map below is an ink bubble drawing by artist **Roland Flexner** (<http://rolandflexner.com>). He created a drawing series by blowing ink-and-soap bubbles onto paper. These bubble painting are worlds within themselves. I was curious to see what kind of map these bubble paintings would create.



Roland Flexner bubble painting (View the map [Open full screen](#) ↗

(<https://tangrams.github.io/spheremap-demos/?url=styles/flexner.yaml#12/37.8773/-121.9290>)

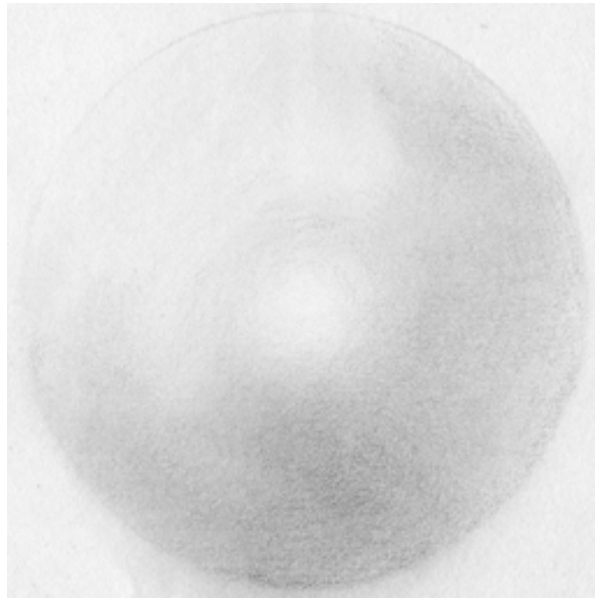
Scientist and artist **Kelsey Brookes** (<http://www.kelseybrookes.com>) paints the invisible worlds of atoms and molecules. Brookes' molecular paintings explore scale from atomic structures up close to resonating and reverberating universes from afar. View the map below created with one of his circular paintings.



Kelsey Brookes mandala (View the map Open full screen ↗ (<https://tangrams.github.io/spheremap-demos/?url=styles/kelseybrookes.yaml#12/37.8773/-121.9290>))

Walkabout Relief

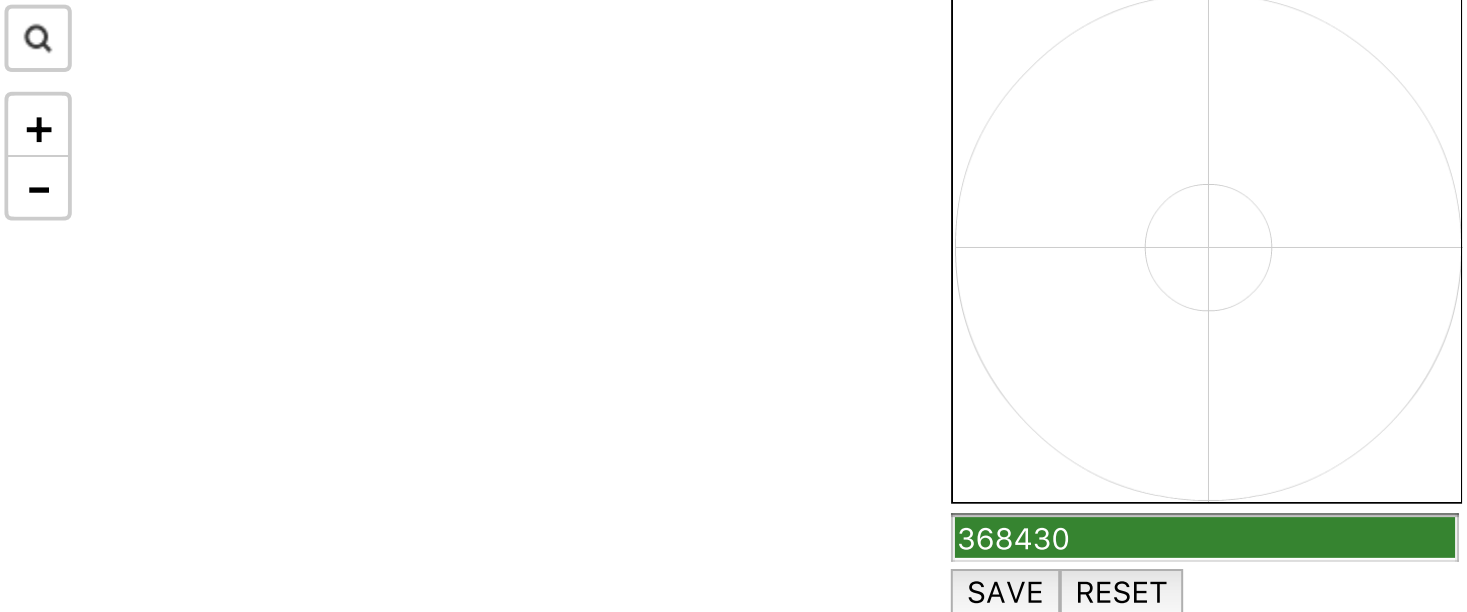
The pencil drawing below is the sphere map we chose for our outdoor style Walkabout. **View the complete style here (<https://tangrams.github.io/walkabout-style-more-labels/#12/37.7859/-122.4440>)**. The map shows solid bright blue for water areas in contrast with the grayscale earth and terrain layer which provides the base upon which all other map layers sit. To learn more about Walkabout cartography, **click here (<https://mapzen.com/blog/walkabout/>)** to see Nathaniel's tour of our outdoor style.



Walkabout Relief Shading (*View the map* **Open full screen** ↗ (<https://tangrams.github.io/spheremap-demos/?url=styles/walkabout.yaml#12/37.8773/-121.9290>))

Sphere Map Demo

Now that you've had a taste of what sphere maps can do, explore and paint your own with this amazing demo Peter built below.



[Leaflet](#) | Geocoding by [Mapzen](#)

Sphere map demo by Peter Richardson

Correction

2016.07.07 - Clarify Walkabout example shows shaded relief only, not full map style.

· 06 July 2016 ·

Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.



© 2017 Mapzen

Closing Mapquest Open Tiles, Opening the Future

[tangram](/tag/tangram) (</tag/tangram>) [mapzen-js](/tag/mapzen-js) (</tag/mapzen-js>)

The Past

Are you seeing maps that look like this?



Six years ago (<http://devblog.mapquest.com/2010/08/24/mapquest-opens-tiles-and-style-enables-rapid-data-updates/>), Mapquest presented Open Tiles to the world. Sadly, Mapquest shut them down on July 11 (<http://devblog.mapquest.com/2016/06/15/modernization-of-mapquest-results-in-changes-to-open-tile-access/>).

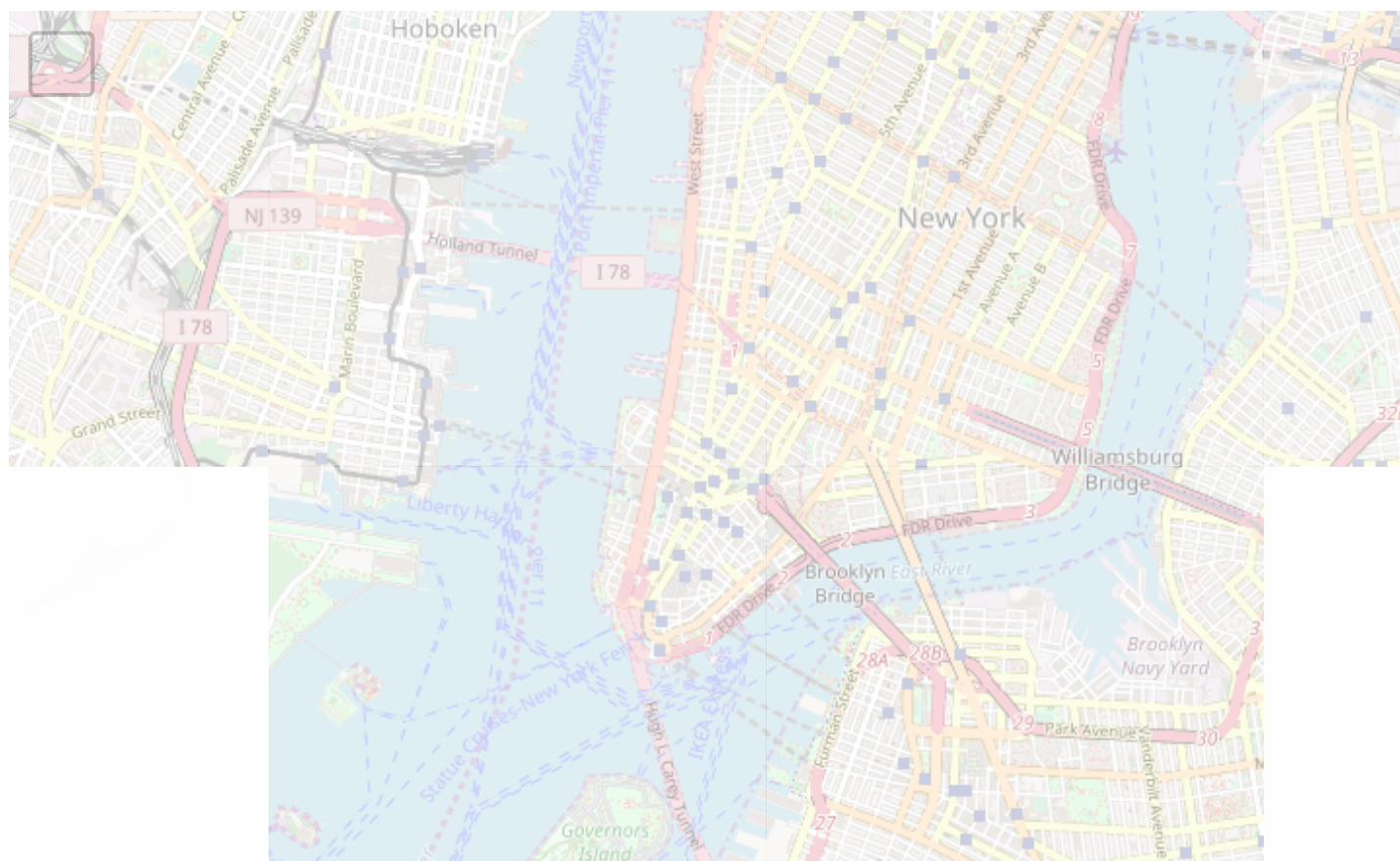
The Present

But as one door closes, others open. Luckily, you now have many alternatives when it comes to making maps based on **OpenStreetMap** (<http://www.openstreetmap.org/>)! Our friends at Stamen provide some of the **nicest looking raster tiles around** (<http://maps.stamen.com/>).



The Future – Get It Today

If you're seeking out maps from the future, we suggest our **Tangram** (<https://mapzen.com/products/tangram/>) mapping engine and our **Vector Tile Service** (<https://mapzen.com/documentation/vector-tiles/>). Tangram uses WebGL to generate great looking maps, whether they are base maps...





or dynamic rendering of cityscapes...



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

Tilt Ikeda, by Patricio Gonzales Vivo [Learn more \(https://mapzen.com/blog/tilting-ikeda/\)](https://mapzen.com/blog/tilting-ikeda/)!

Getting Started

Check out the **Tangram documentation** (<https://mapzen.com/documentation/tangram/>), learn **how to get started** (<https://mapzen.com/documentation/tangram/walkthrough/>), and take a look at some **more advanced examples** (<https://github.com/tangrams/WeatherNow/blob/gh-pages/README.md>).

But we always want to make things easier for our friends, so we are testing a **mapzen.js library** (<https://github.com/mapzen/mapzen.js>). It will let you make a Leaflet map based on Mapzen styles and add a geocoder in just a few lines of code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1"/>
    <link rel="stylesheet" href="https://mapzen.com/js/mapzen.css">
    <script src="https://mapzen.com/js/mapzen.min.js"></script>
  </head>
  <body>
    <div id="map"></div>
    <script>
      // Add a map to the #map DIV, and center it on New York:
      var map = L.Mapzen.map('map');
      // Set default view on New York at zoom level 13
      map.setView([40.70531, -74.009], 13);
      var geocoder = L.Mapzen.geocoder('search-api-key');
      geocoder.addTo(map);
    </script>
  </body>
</html>
```

(In fact, we used `mapzen.js` to make the base map above.)

It's a 0.1.0 product, so **please let us know** (<https://github.com/mapzen/mapzen.js/issues>) if you see any issues or have any suggestions!

You can also make changes to a Tangram map on the fly with the **beta verison of Tangram Play** (<https://mapzen.com/tangram/play/>). (You can learn more about **YAML** (<https://mapzen.com/documentation/tangram/yaml/>) and **what makes a scene file tick** (<https://mapzen.com/documentation/tangram/Scene-file/>).)

Let us know (<http://twitter.com/mapzen>) what maps you make!

· 12 July 2016 ·

John Oram



Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

Concordances with Wikipedia data

data (/tag/data) **whosonfirst** (/tag/whosonfirst)



*Illustration: Olga Kavvada. Photo Credits: **Wikipedia** (<https://www.wikipedia.org/>), **Stay Connected** (<https://theinnovationenterprise.com/summits/global-sports-innovation-summit-boston/stay-connected>)*

We recently **tweeted** (<https://twitter.com/alloftheplaces/status/748202320677109760>) that Who's On First was “holding hands” with even more of Wikipedia. Today I will talk about how that work was completed.

To recap:

- **155,000** more Who's on First records were linked up to Wikipedia
- **2 million** more localized names (in multiple languages) were added for **135,000** Who's On First records

Our vision is to be able to make a unique connection between our **Who's On First** (<https://whosonfirst.mapzen.com/>) data and the corresponding Wikipedia page. Who's On First is Mapzen's gazetteer of places, each record has a stable identifier and descriptive properties about that place. We use Who's On First data for several Mapzen services to help with **labelling** the map and improving **search** results.

Wikipedia is an encyclopedia which is collaboratively built by its users and has more than 5,000,000 articles covering many subjects and disciplines. The benefit of Wikipedia is that it has come to be one of the most popular websites on the internet, constantly evolving and one of the largest general reference works. Linking Who's On First data to Wikipedia articles, allowing the two projects to "hold hands", enables us (and you!) to benefit from this collective knowledge.

Wikipedia Structure

The first step is to understand the structure of Wikipedia's database. Wikipedia has a web service API that provides access to its main wiki features, data and metadata, namely the **MediaWiki API** (https://www.mediawiki.org/wiki/API:Main_page). An **API** (<http://readwrite.com/2013/09/19/api-defined/>) is essentially a tool that allows software applications to talk to each other in a structured way.

To make things a little more helpful (or complicated!), Wikipedia has a related project called **Wikidata** (https://www.wikidata.org/wiki/Wikidata:Main_Page) that acts as a central storage space for all the structured data in Wikipedia. Each Wikidata record is uniquely identified and links to each related Wikipedia entry (think *web page*) in all the different Wikipedia's languages (282 at last count!) for that entity.

For example, in the English language Wikipedia the page is **Spain** (<https://en.wikipedia.org/wiki/Spain>) and in the Spanish language Wikipedia the page is **España** (<https://es.wikipedia.org/wiki/Espa%C3%B1a>). As a result, in order to find the specific country (or city!) in the Wikipedia of your choice, you would need to know the language code of the Wikipedia site along with the title of the page in that language.

Since we're interested in adding *more* localized names in all the languages to each Who's On First record, we'll look for the feature first by the name we already know it as, then collect and add the rest of the names from the many Wikipedias keyed off the Wikidata ID identifier. For Spain the Wikidata ID is **Q29** (<http://www.wikidata.org/wiki/Q29>) and that record has entries for both of the articles mentioned above, and in many more languages. We're lucky each project has unique identifiers for places!

Wikipedia Titles

To link up the projects, we needed to get the original Wikipedia titles of each Who's On First record (for example the borough that is called `Bronx` in Who's On First but named `The Bronx` in Wikipedia and the `Queenstown` locality in Who's On First is named `Queenstown, New Zealand` in Wikipedia), the Wikidata ID of each entry, the Wikipedia url and all the localized language translations of each place. In addition, we wanted to get population data for the

administrative places, elevation from the sea level, area and latitude and longitude coordinates. The entire code structure can be found in this **repository** (<https://github.com/mapzen-data/wikipedia-notebooks>) as an iPython Notebook.

The first step of adding concordance between Who's On First and Wikipedia data was to identify a connection point. Wikipedia allows you to search its database for a place name and returns the most likely Wikipedia page title. We used this API query to identify the potential original Wikipedia page title for all the administrative places in our database using the Who's On First name as the key input argument.

A sample python request using the `requests` python package for the original Wikipedia titles (`wk:page`) is shown below:

```
request_API = ("https://en.wikipedia.org/w/api.php?action=query&list=search&srsearch=%s&srresult_request=requests.get(request_API)
```

This worked relatively well but as you can imagine for the ambiguous cases (where multiple results could match) Wikipedia sometimes returned page titles that were not the ones we were looking for. We looked through the results and applied several cleanup steps to identify which ones were correct and which ones were not.

In the table below you can see a selected subset of the raw results that we got from Wikipedia `wk:page` for each Who's On First `wof:name`.

	wof:name	wk:page
3	South America	South America
32	Rajshahi Airport	List of airports in Bangladesh
39	Tafaraoui Airport	Oran Tafraoui Airport
49	Laghouat Airport	L'Mekrereg Airport
257	Sky Harbor Residential Airpark	Hernando Village Airpark
517	Marked Tree Municipal Airport	Arkansas Highway 308
905	Junction Airport	Grand Junction Regional Airport
1279	La Fonda Ranch Airport	Once Upon a Time in the West
73424	Woods of Eden Rock	Rock music
154819	Bronx	The Bronx

We used several approaches to sanity check the results and only keep the correct ones to prevent importing bogus Wikipedia concordances into the Who's On First database. This involved a classification process to evaluate if the result name from Wikipedia was a good match for the input name from Who's On First (`correct` column). The entire code for the data clean-up can be found in this **iPython notebook** (https://github.com/mapzen-data/wikipedia-notebooks/blob/master/Jupyter_notebooks_with_analysis/Find_original_wikipedia_title_and_wordcount.ipynb).

The first approach involved identifying and discarding any Wikipedia titles that were included in a “blacklist”. This blacklist consisted of page titles that included numbers `0` thru `9` or any of the words `timeline` , `birthday` , `political` , `environmental` or `music` which would probably point to aggregate Wikipedia pages that were not of interest to us. This would help eliminate false Wikipedia concordances such as the ones listed below:

	wof:name	wk:page	correct
29	Dhaka Tejgaon Airport	List of airports in Bangladesh	NO
123	Flying Crown Airport	List of airports in Canada (A–B)	NO
164	Wawasee Airport	1974 Super Outbreak	NO
1345	City of Friendship Airport	National Airlines (1934–1980)	NO
40303	Amado Bahia	Timeline of Salvador, Bahia	NO
78901	Grafs First	Rolf Graf (musician)	NO

Another easy fix was to try and find results that were not referring to the same placetype even though their names might match. Such examples would be entries that did not have the word `Airport` or `Facility` in both the input name as well as the returned Wikipedia page. On the other hand, if the words `District` or `Municipality` were included in the returned result, they were classified as correct as we were looking for administrative places. Wikipedia sometimes returned the disambiguation page as a page title result which was disregarded.

	wof:name	wk:page	correct
37	Tinfouchy Airport	Tinfouchy	NO
2317	Logan County Airport	Logan Airport (disambiguation)	NO
13206	Dolní Újezd	Dolní Újezd (Svitavy District)	OK
229925	Ribnica	Ribnica, Ribnica	OK

For the remaining results, we calculated the **Levenshtein distance** (https://en.wikipedia.org/wiki/Levenshtein_distance) of the input name and the result name which is a metric for quantifying the difference between two **strings** ([https://en.wikipedia.org/wiki/String_\(computer_science\)](https://en.wikipedia.org/wiki/String_(computer_science))). This value would be used as a metric for dissimilarities between the two names. To avoid misclassifying occurrences that involved some kind of hierarchical order (for example, `Scottsville` vs. `Scottsville, Kentucky`), we calculated a separate the Levenshtein distance between the Wikipedia result and the name after joining with its corresponding region or country.

The Levenshtein distance metric provided insight on which results were probably correct, thus their names were as similar as possible. We allowed entries that matched 100% with the Levenshtein distance metric but we were flexible enough to accept up to 30% dissimilarities (see `Tafaraoui Airport` example). Dissimilarities more that 80% of the strings were marked as not correct. Below you can identify some examples that were classified with the use of the Levenshtein distance.

	wof:name	wk:page	name_region	leve_dist_r	leve_region_r	leve_country_r	correct
28	Coxs Bazar Airport	Cox's Bazar Airport	Chittagong	0.05	0.43	0.43	OK
31	Ishurdi Airport	Ishwardi Airport	Rajshahi	0.12	0.48	0.52	OK
39	Tafaraoui Airport	Oran Tafraoui Airport	Oran	0.29	0.52	0.58	OK
65	In Anguel Airport	Tamanrasset	Tamanghasset	0.88	0.68	0.88	NO
73	Tiska Airport	Djanet Inedbirene Airport	Illizi	0.64	0.88	0.84	NO
109	Lewis B Wilson Airport	Middle Georgia Regional Airport	Georgia	0.61	0.90	0.86	NO
160	Pontiac Municipal Airport	Oakland County International Airport	Illinois	0.58	0.86	0.88	NO
1133	Scottsville	Scottsville, Kentucky	Kentucky	0.48	0.00	0.42	OK
2864	Scottsville	Scottsville, California	California	0.52	0.00	0.46	OK
4891	Scottsville	Scottsville, Kansas	Kansas	0.42	0.00	0.42	OK

Each of the Who's On First records has an associated placetype that describes its hierarchy. A semi-automatic classification involved checking for placetypes in the Who's On First data and the results from the Wikipedia page. The placetypes between Who's On First and Wikipedia should match else the Wikipedia title was considered as wrong. In some cases multiple Who's On First entries would share a name but have different placetypes associated with them (see example for `China` in the table below). For those cases the Wikipedia title was connected to the Who's On First entry with the higher ranking of a placetype in the hierarchy of places (for example, `country > locality > neighborhood`).

	wof:name	wk:page	placetype	correct
12	China	China	country	OK
153054	China	China	county	NO
184843	China	China, Nuevo León	locality	OK
208292	China	China, Texas	locality	OK

The final technique was to manually classify the entries as correct or not by going through the ones not yet classified especially in the areas where we had personal knowledge. The Slavic and Greek languages were classified by hand as it was impossible to find a point of connection between the different alphabets.

A snippet of the Wikipedia results and our final quality classification is shown in the table below:

	wof:name	wk:page	correct
3	South America	South America	OK
32	Rajshahi Airport	List of airports in Bangladesh	NO
39	Tafaraoui Airport	Oran Tafraoui Airport	OK
49	Laghouat Airport	L'Mekrereg Airport	maybe
257	Sky Harbor Residential Airpark	Hernando Village Airpark	maybe
517	Marked Tree Municipal Airport	Arkansas Highway 308	NO
905	Junction Airport	Grand Junction Regional Airport	maybe
1279	La Fonda Ranch Airport	Once Upon a Time in the West	NO
73424	Woods of Eden Rock	Rock music	NO
154819	Bronx	The Bronx	OK

After cleaning up the data gathered from Wikipedia we ended up with about **155,000** of entries with Wikipedia titles classified as correct (OK in the table above), **24,000** were uncertain (maybe), and **81,000** were classified as wrong (NO).

Wikidata IDs

Having identified the correct Wikipedia page title for most of our entries we then requested the Wikidata ID for each page in our dataset. This returned a unique identifier for each entry that had an original Wikipedia title. These values are extremely useful as they provide a point of connection between all the different Wikipedias in the 282 different languages where the data is stored in a structured way.

A sample python request using the `requests` python package for the Wikidata IDs (`wd:id`) is shown below:

```
request_API = ("https://en.wikipedia.org/w/api.php?action=query&prop=pageprops&titles=%s&result_request=requests.get(request_API)
```

We also requested the word count of each web page from the Wikipedia API. This will help generate a quantifiable metric for assessing the importance of each place and will be described in a future post. A sample python request using the `requests` python package for the `wordcount` is shown below:

```
request_API= ("https://en.wikipedia.org/w/api.php?action=query&list=search&srsearch=%s&srresult_request=requests.get(request_API)
```

A subset of our Who's On First data with the corresponding Wikidata IDs is shown below:

	wof:id	wof:name	wk:page	wd:id
0	102191569	Asia	Asia	Q48
17	421166797	Georgia	Georgia	Q4962
18	421180189	Somalia	Somalia	Q1045
19	421185849	Kazakhstan	Kazakhstan	Q232
21	421202109	San Marino	San Marino	Q238
25	421205775	Staten Island	Staten Island	Q18432
100	102521867	Brooks Air Force Base	Brooks Air Force Base	Q4974968
103	102521877	Los Angeles Air Force Base	Los Angeles Air Force Base	Q6681941

Wikipedia languages

Another important feature we were interested in getting from Wikipedia was different names in many languages for each Who's On First record. Wikipedia has been designed in many different languages and also gives aliases to names for localized languages. This information is be valuable for labeling places on a map and for search engines.

The average number of language aliases for a Wikipedia place was 15 and some places had up to 266. Because some entries had so many Wikipedia aliases several consecutive API calls had to be made to get all the aliases for each place as each request only returns a small number of

aliases at a time.

A sample python request using the `requests` python package for the languages is shown below:

```
request_API= ("https://en.wikipedia.org/w/api.php?action=query&titles=%s&prop=langlinks&format=json" % title)
result_request=requests.get(request_API)
```

Here are a few of Yosemite Valley's different language aliases:

```
en : Yosemite Valley
nl : Yosemite Valley
pt : Yosemite Valley
hr : Yosemite Valley
sr : Јосемити Вали
ca : Yosemite Valley
ur : یوسمتھی ویلی، کیلی فورنیا
sh : Yosemite Valley
new : योसेमाइट भ्याली
vo : Yosemite Valley
es : Yosemite Valley
```

Wikipedia demographics and location data

For administrative places, it would be great if we could add more information on population, elevation from sea level and location coordinates. Since this data cannot be directly accessed from the MediaWiki API, we used Wikidata's **SPARQL API** (<https://query.wikidata.org/>) query service. By using SPARQL we were able to get population, elevation and location data for some administrative places in Wikipedia.

The bottleneck of this process is that Wikipedia categories are not well defined so it is hard to search for all administrative places as they can be under many different categories. We searched for the most prominent ones like `country` , `region` , `county` , `city` , `town` , `village` , `neighborhood` , `airport` and `archaeological site` and then by using the Wikidata IDs joined them to the administrative places in Who's On First.

Using this technique we were able to add population data to about **5,500** records in Who's On First, elevation to **10,000** and latitude and longitude to about **26,000**.

	wof:id	wof:name	wk:page	wd:id	population	elevation_above_sea_level	lat	lon
3382	101773037	Eibar	Eibar	Q496567	27414	121	43.184284	-2.473275
3573	101774153	Berga	Berga	Q15487	16238	704	42.100000	1.845556
3752	101775359	Almoharín	Almoharín	Q1630552	1933	307	39.176366	-6.044045
8743	101820845	Falkenberg/Elster	Falkenberg/Elster	Q570094	6529	86	51.583056	13.232222
8948	101823963	Chmielnik	Chmielnik	Q991973	3891	240	50.611667	20.749722
9840	101830961	Arenas de Mar	Punta Arenas	Q51599	127454	34	-53.150000	-70.916667
30006	101965793	Pindorama	Pindorama	Q1648152	13109	527	-21.185833	-48.906944
30012	101965879	Tremembé	Tremembé	Q1772599	34823	560	-22.957778	-45.548889
30046	101966387	Sana	Sana'a	Q2471	2957000	2150	15.350000	44.200000

We love our data! (but it can always be improved...)

Wikipedia is a huge source of information and we are proud to have the Who's On First gazetteer "holding hands" with more of it.

Of the **155,000** Who's on First records that were linked up with Wikipedia, almost all Wiki places were in multiple languages. This allowed us to add nearly **2 million** localized names for **135,000** Who's On First records. A smaller set of records received some additional properties. For example, see **Italy**

(<https://whosonfirst.mapzen.com/spelunker/id/85633253/#5/41.546/12.560>), where all the localized names are under the tab `names` and the Wikipedia concordances under `wof - concordances`.

You can see the full effects of the new Wikipedia data in Who's on First by using the **Spelunker** (<https://mapzen.com/blog/spelunker-jumping-into-who-s-on-first/>) to view lists of all the Who's On First places with:

- **Wikidata concordances**
(<https://whosonfirst.mapzen.com/spelunker/concordances/wikidata/>)
- **Wikipedia concordances**
(<https://whosonfirst.mapzen.com/spelunker/concordances/wikipedia/>)

This is still a work in progress and as new data comes into Who's On First we will need to keep our Wiki concordances fresh.

We wish to use this data for other types of analyses as well, such as a ranking method of feature importance to help mapping and search. We will keep you posted!



Olga Kavvada

Former Mapzen data team intern. Passionate about the outdoors and all things spatial.

© 2017 Mapzen

Large-scale Map Matching with Meili

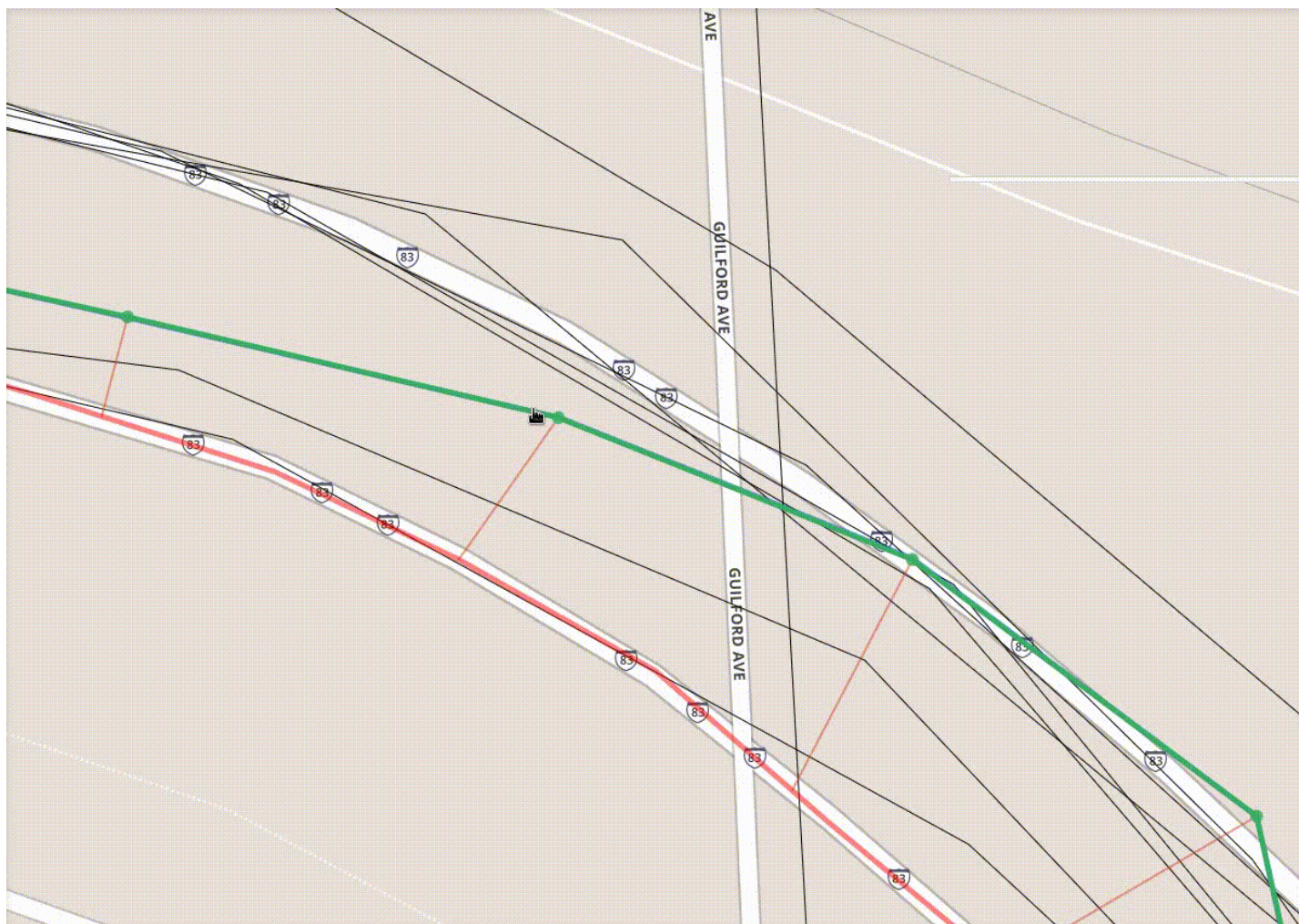
engineering (/tag/engineering) **routing** (/tag/routing)

This is a joint post by Kevin Kreiser (Mapzen) and Tao Peng (Mapillary).

The world is full of uncertainties. When a Mapillary member uploads his or her favorite street sequence, technically we do not know which streets (in the case of OpenStreetMap - “ways” to be more specific) are mapped. This is because the sequence is usually noisy, and rarely properly aligned with the streets. In order to know what street segments you are mapping, we need a GIS technique called **map matching** (https://en.wikipedia.org/wiki/Map_matching).

Meili – a collaboration between Mapillary and Mapzen

We are pleased to announce our map matching solution, Meili, a collaborative **open source project** (<https://github.com/valhalla/Meili>) between Mapillary and team Valhalla. Meili is built as a component of Valhalla, an open source routing engine for use with OpenStreetMap data. In Valhalla, Meili focuses on providing a state-of-the-art map matching solution as a service and library. As Meili is built with Valhalla, it shares the awesomeness of Valhalla: open data, small memory footprint and multiple transport modes (so you can match traces of different travel modes).



Meili can easily get thousands of points matched in a second.

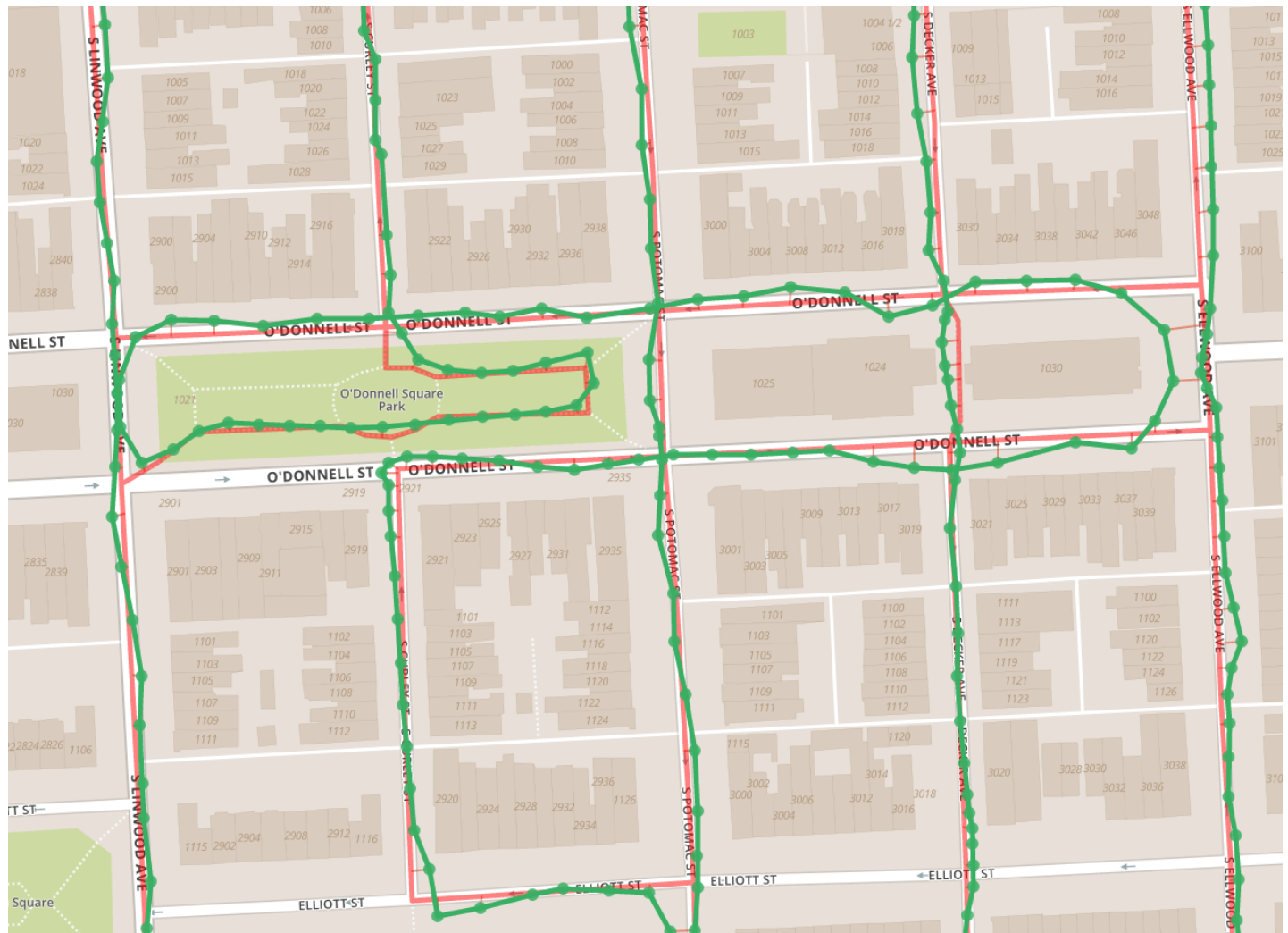
High performance map matching

We built Meili with performance in mind, to be able to process large-scale spatial data. At Mapillary, Meili can get all street photos, over 70 million points, matched to OpenStreetMap streets in a few hours. This allows us to do a complete matching every week or so, to keep all photos associated with the latest OpenStreetMap data.

How we use Meili at Mapillary

Associating photos and OpenStreetMap ways can benefit both Mapillary and the OpenStreetMap community. For Mapillary, in addition to properly aligning the green coverage lines on the map, the logical association can tell us how many photos have mapped a street segment and thus help us decide if we should further map it or not. For the OpenStreetMap

community, information such as traffic signs and road conditions extracted from the street photos are also associated with OpenStreetMap ways. This can help detect issues and add road features to OpenStreetMap road network.



A long walk sequence (green line) by **@tallguy** (<http://www.mapillary.com/profile/tallguy>) is matched to streets (red lines). Follow along in the photos below.

□

General applications

Meili can be a useful tool for many applications involved with large-scale spatial data. Location-based services can use it to process location-based activities. Traffic researchers can use it to process floating car data for traffic analysis. If you need to deal with noisy spatial data and you want to correlate that with OpenStreetMap, Meili will be a good choice.

Future

In the future we will continue to work closely with team Valhalla to better integrate Meili with Valhalla and provide more features such as online map matching, which is a key component for car navigation. We will also continue to make it more accurate and faster for large-scale map matching. And of course, we will build more applications based on map matched street photos.

If you want to join us, you're welcome to **fork Meili** (<https://github.com/valhalla/Meili>), **try it** (https://github.com/valhalla/Meili/blob/master/docs/run_service_in_docker.md), or **open issues** (<https://github.com/valhalla/Meili/issues>).

· 14 July 2016 ·



Tao Peng

Mapillary map matching master.

Kevin Kreiser



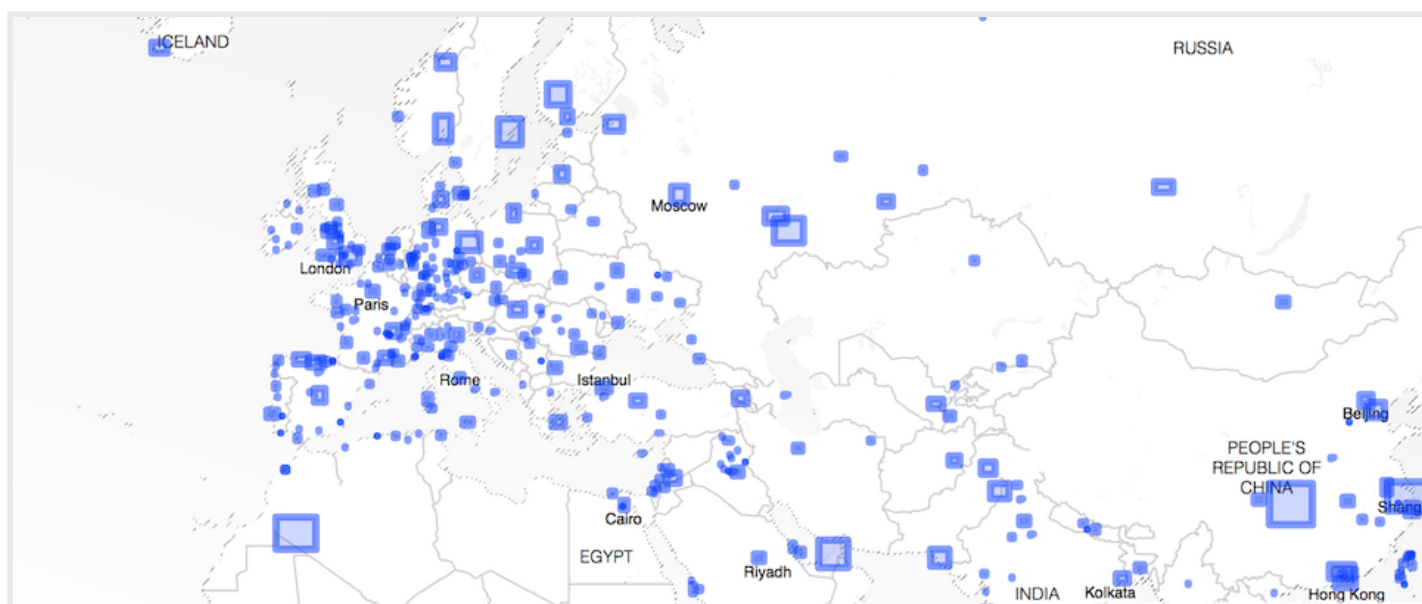
Kevin works on routing at Mapzen but secretly tries to work in all mapping disciplines. Er iss aa Pennsilfaanisch Deitscher.

© 2017 Mapzen

Metro Extracts On Demand

data (</tag/data>)

Metro Extracts (<https://mapzen.com/metro-extracts/>) is getting some work done, and it means more easy-to-use free open geodata for you, you, and you. **Try out a new on-demand feature** (<https://mapzen.com/data/metro-extracts-alt/>) and **let us know if you're available for an interview** (<http://goo.gl/forms/OIEdDW7RIC04oqo2>) so we know it's working.

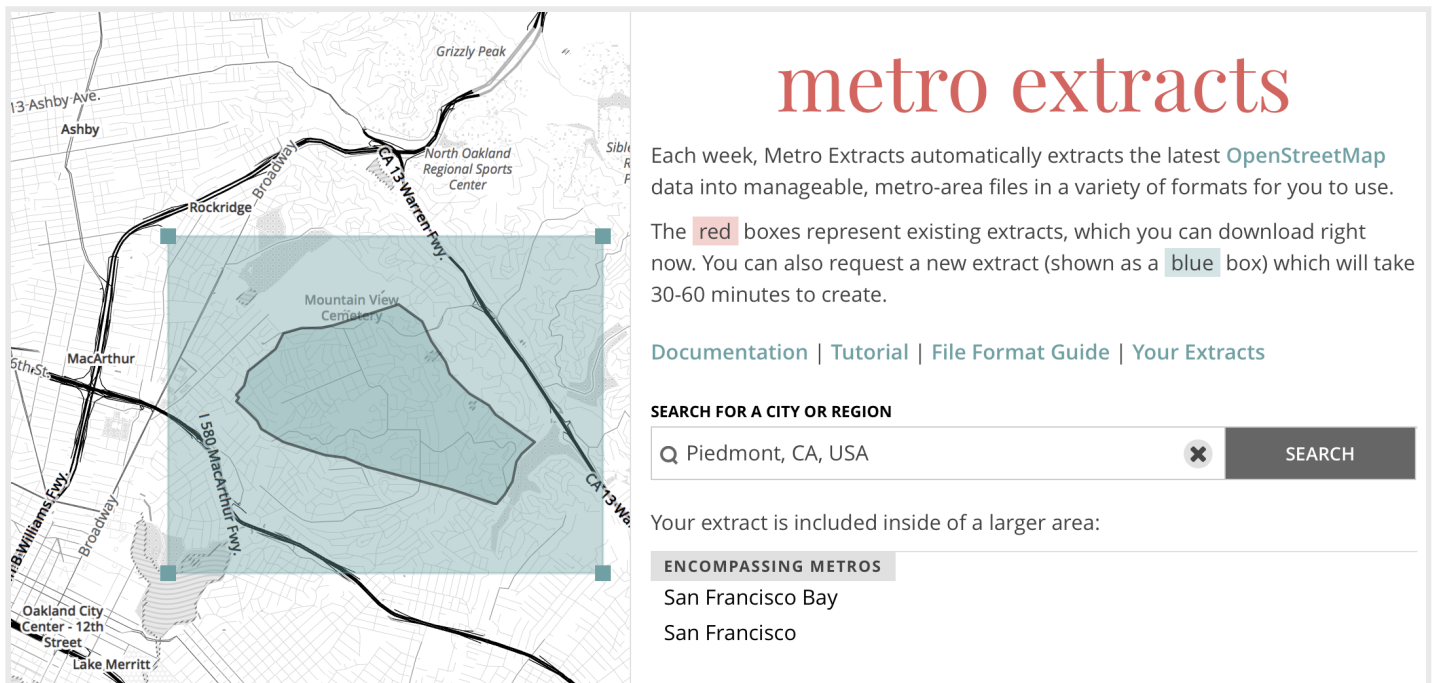


Not a lot has changed since **Ingrid wrote about Metro Extracts** (<https://mapzen.com/blog/metro-extracts-101/>) way back in 2014: it's still providing free, easy-to-use slices of OpenStreetMap data centered on major metropolitan areas, it's getting updated every week, and we're still accepting requests for new areas via Github pull requests all the time. The last part has always been a bummer, though. In order to get data for a place we don't know about, you need to know your way around a directed acyclic graph of geographic bounding boxes and have the patience to wait for the once-weekly process to get around to your request.

Today, we're opening up a new feature: your own custom extracts, the ability to search for a place with **Mapzen Search** (<https://mapzen.com/products/search/>), request an area of OpenStreetMap data thanks to **our worldwide gazetteer Who's On First**

(<https://whosonfirst.mapzen.com>), and get the results within an hour (give or take). It's all part of a interface update we're doing for Metro Extracts that you can check out at mapzen.com/data/metro-extracts-alt (<https://mapzen.com/data/metro-extracts-alt/>).

We're still publishing 200 of the world's most popular major metro areas **as I suggested in 2011** (<http://mike.teczno.com/notes/piecemeal-geodata.html>) at *State Of The Map* in Denver. We've listened to your feedback, so now you can pick out data for towns and places important to you as well.



The screenshot shows the 'metro extracts' interface. On the left is a map of the San Francisco Bay Area with a light blue rectangular region highlighted. Labels on the map include 'Grizzly Peak', 'North Oakland Regional Sports Center', 'Mountain View Cemetery', 'MacArthur', 'Oakland City Center - 12th Street', 'Lake Merritt', '13 Ashby Ave.', 'Ashby', 'Rockridge', 'Broadway', '15th St', '18th St', '19th St', '20th St', '21st St', '22nd St', '23rd St', '24th St', '25th St', '26th St', '27th St', '28th St', '29th St', '30th St', '31st St', '32nd St', '33rd St', '34th St', '35th St', '36th St', '37th St', '38th St', '39th St', '40th St', '41st St', '42nd St', '43rd St', '44th St', '45th St', '46th St', '47th St', '48th St', '49th St', '50th St', '51st St', '52nd St', '53rd St', '54th St', '55th St', '56th St', '57th St', '58th St', '59th St', '60th St', '61st St', '62nd St', '63rd St', '64th St', '65th St', '66th St', '67th St', '68th St', '69th St', '70th St', '71st St', '72nd St', '73rd St', '74th St', '75th St', '76th St', '77th St', '78th St', '79th St', '80th St', '81st St', '82nd St', '83rd St', '84th St', '85th St', '86th St', '87th St', '88th St', '89th St', '90th St', '91st St', '92nd St', '93rd St', '94th St', '95th St', '96th St', '97th St', '98th St', '99th St', '100th St'. On the right, the title 'metro extracts' is in red. Below it, text explains that Metro Extracts automatically extracts the latest OpenStreetMap data into manageable, metro-area files. It mentions that red boxes represent existing extracts and blue boxes represent new requests. There are links for 'Documentation | Tutorial | File Format Guide | Your Extracts'. A search bar contains 'Piedmont, CA, USA' and a 'SEARCH' button. Below the search bar, it says 'Your extract is included inside of a larger area:' and lists 'ENCOMPASSING METROS' as 'San Francisco Bay' and 'San Francisco'.

The process is all thanks to a new data product created by Mapzen, called On-Demand Extract Service (ODES). With a free Mapzen developer account, you can search for places around the world and create custom excerpts of OpenStreetMap data useable in popular geospatial tools:

- ESRI shapefiles with raw OSM data and thematic layers **via Imposm** (<https://imposm.org>).
- GeoJSON data compatible with all kinds of web map platforms.
- Land and water areas generated from OpenStreetMap coastlines.

This is all hot off the press, and we'd love your help testing the new service. Try it out at mapzen.com/data/metro-extracts-alt (<https://mapzen.com/data/metro-extracts-alt/>), and **get in touch to participate in user research to make it better** (<http://goo.gl/forms/OIEdDW7RIC04oqo2>).



Michal Migurski

Oakland-dwelling geodata cyclist.

© 2017 Mapzen

Augmenting reality with open software and data

data (/tag/data) **tangram** (/tag/tangram)

Some of us at Mapzen begrudgingly accept that a lot of the 1990s happened 20 years ago, and today it seems almost quaint that things went viral before the Internet was widespread. However, it's not particularly surprising that the old-school Pokémon cards and video games of the mid-90s made the jump to augmented reality and virality in big way.

Combining AR games and check-in apps with a cultural touchstone seems obvious. But beyond the game play, what do you need to make the map in the style of Pokemon Go? In short, vector map data, map display tools, geolocation, and lots of points of interest.



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

(Thanks for colors, Patricio!)

Click to tilt and zoom in NYC! ([https://tangrams.github.io/tangram-frame/?url=https://gist.githubusercontent.com/burritojustice/94d9101404958df5b68985da13ca3e9d/raw/POIkemon.yaml#16/40.74454/-73.98745](https://tangrams.github.io/tangram-frame?url=https://gist.githubusercontent.com/burritojustice/94d9101404958df5b68985da13ca3e9d/raw/POIkemon.yaml#16/40.74454/-73.98745))

url=https://gist.githubusercontent.com/burritojustice/94d9101404958df5b68985da13ca3e9d/raw/POIkemon.yaml#16/40.74454/-73.98745)

Click to see SF (<https://tangrams.github.io/tangram-frame?url=https://gist.githubusercontent.com/burritojustice/94d9101404958df5b68985da13ca3e9d/raw/POIkemon.yaml#16/37.79287/-122.39425>)

url=https://gist.githubusercontent.com/burritojustice/94d9101404958df5b68985da13ca3e9d/raw/POIkemon.yaml#16/37.79287/-122.39425)

So what's going on in this map? You can take a look **in Tangram Play**

([https://mapzen.com/tangram/play/?](https://mapzen.com/tangram/play/?scene=https://gist.githubusercontent.com/burritojustice/94d9101404958df5b68985da13ca3e9d/raw/POIkemon.yaml#17/40.74784/-73.98845)

scene=https://gist.githubusercontent.com/burritojustice/94d9101404958df5b68985da13ca3e9d/raw/POIkemon.yaml#17/40.74784/-73.98845).

import

```
import:
  - https://tangrams.github.io/blocks/global.yaml
  - https://tangrams.github.io/blocks/geometry/tilt.yaml
  - https://tangrams.github.io/blocks/geometry/rotation.yaml
  - https://tangrams.github.io/blocks/lines/glow.yaml
  - https://tangrams.github.io/blocks/points/glow.yaml
  - https://tangrams.github.io/blocks/polygons/pixelate.yaml
```

Here we're importing **blocks** (<http://tangrams.github.io/blocks/>) of functionality. You can read more about **Tangram Blocks** (<http://tangrams.github.io/blocks/>) here, but the goal is to import interesting modules and keep your scene file from getting huge and complicated. You can also import an existing scene file, like Refill or Zinc and draw things on top of it – more details are available in the **Tangram docs** (<https://mapzen.com/documentation/tangram/import/>).

textures


```
textures:
  pois:
    url: https://raw.githubusercontent.com/tangrams/refill-style-more-labels/gh-pages,
    filtering: mipmap
    sprites:
      # define sprites: [x origin, y origin, width, height]
      airport: [870, 0, 38, 38]
      aquarium: [732, 168, 38, 38]
      art-gallery: [640, 168, 38, 38]
      athletics-sports: [184, 168, 38, 38]
      atm: [918, 126, 38, 38]
      automotive-shop: [0, 168, 38, 38]
      bakery: [548, 168, 38, 38]
      bank: [964, 126, 38, 38]
      bar: [230, 168, 38, 38]
```

Tangram can work with images as well as vectors. (Remember **sphere maps** (<https://mapzen.com/blog/sphere-maps/>)?) Textures and materials are cool and crazy – **you can** (<https://mapzen.com/documentation/tangram/Materials-Overview/#textures>) **learn more about them** (<https://mapzen.com/documentation/tangram/textures/>) in the documentation – but in this case we are cropping POI icons out of a larger image in one of the Mapzen house styles.

styles

Styles (<https://mapzen.com/documentation/tangram/Styles-Overview/>) is where you tell polygons, lines, points, text what to do.

```
water:
  base: polygons
  mix: [geometry-rotation, geometry-tilt, polygons-pixelate]
earth:
  base: polygons
  mix: [geometry-rotation, geometry-tilt, polygons-pixelate]
buildings:
  base: polygons
  mix: [geometry-rotation, geometry-tilt]
buildings_outline:
  base: lines
  mix: [geometry-rotation, geometry-tilt, lines-glow]
points:
  base: points
  mix: [geometry-rotation, geometry-tilt]
icons:
  base: points
  texture: pois
  interactive: True
  mix: [geometry-rotation, geometry-tilt, points-glow]
```

Remember the `import` section? Here we are referencing them to make the polygons, lines and points do fun things like tilt, rotate, pulse and/or have a cool pixellated appearance. (If you comment out the `mix` line in roads, you'll quickly see them stop rotating and tilting and, um, depixelate.)

We can also manipulate shaders here – in the `geometry-tilt` and `geometry-rotation` blocks, we are telling WebGL to tell your graphics card rotate and tilt between certain map zoom levels. You can more shader instructions in the **geometry-rotation Tangram block** (<https://tangrams.github.io/blocks/geometry/rotation.yaml>) – we could have added this into the scene file, but pulling it in as a block keeps the scene file less cluttered.

layers

```
layers:
  water:
    data: { source: mapzen }
    draw:
      polygons:
        style: water
        order: global.order
        color: [0.000, 0.651, 0.760]

  earth:
    data: { source: mapzen }
    draw:
      polygons:
        style: earth
        order: global.order
        color: [0.367, 0.610, 0.514]

  landuse:
    data: { source: mapzen }
    draw:
      polygons:
        style: landuse
        order: global.order
        color: [0.022, 0.755, 0.426]
```

Here we're telling Tangram where to get the data from (in this case, the vector tiles, though this could also be **your own un-tiled geojson files**

(https://mapzen.com/documentation/tangram/sources/#generate_label_centroids)), what colors things should be, and what order they should be in. **Much much more about layers here** (<https://mapzen.com/documentation/tangram/Filters-Overview/#layer-filters>).

The `pois` and `beer` section is an interesting one. We are filtering out points from the vector tiles and looking for the ones that have names.

```

pois:
  data: { source: mapzen }
  filter: { $geometry: [point], not: { kind: [pub, bar] } }
  has-name:
    filter: { name: true }
    # match 1:1 correlations between data and sprite name
  direct-match:
    filter: { area: false }
    draw:
      icons:
        color: [0.255,0.976,1.000]
        sprite: function() { return feature.kind; }

beer:
  data: { source: mapzen, layer: pois }
  filter: { kind: [pub, bar] }
  draw:
    beer-icons:
      #color: red
      color: [.5.,0.9,0.400]
      sprite: function() { return feature.kind; }

```

We're referencing that `icons` style (which references that `pois` texture) to draw the icons, color them, and in the case of one that are bars, make the icons pulse.

Niantic seems to be using Google map data for streets and building geometry, and points of interest (POIs) that they collected from their previous games. However, there are issues with this data: Some of the POIs are already out of date, and coverage is spotty in rural areas. Open map data could make an AR game like Pokémon even better. OpenStreetMap and our Who's On First gazetteer would allow an endless supply of POIs that could be updated more readily, and maintain the novelty of discovery within any AR game.

We used OSM POIs in this map, but we could have easily pulled in other information. Opening the data could expand the game and the user base: imagine using **GTFS transit data** (<https://transit.land/documentation/datastore/api-endpoints.html>) to assign pokémon to bus stops, requiring that you had to take any eggs on bus rides to hatch. You could even use routing tools to determine the best paths between pokéstops and gyms.

We will leave it to you to come up with viral content, but rest assured that the **mapping technology and POI data** (<https://mapzen.com/>) behind any AR app is well within your reach!

· 15 July 2016 ·



John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

Speed Tiles and Traffic-Influenced Routing

routing (/tag/routing)

Valhalla, the open source routing engine which powers our Turn-by-Turn service, was designed to be extensible. Something we've wanted to add is traffic-influenced routing -- but how would this work? We've developed a proof-of-concept where we import "speed tiles" linked to OSM street IDs and use Valhalla's dynamic costing to route around slow roads.

Traffic-Influenced Routing – Proof of Concept

The tiled routing graph design and dynamic costing methods used by Valhalla should readily support integration of both real-time and historical traffic or speed information. This paper describes a proof of concept that was developed to demonstrate traffic-influenced routing. The proof of concept allows entry of speeds for a set of OpenStreetMap (OSM) ways.

These ways are then correlated with Valhalla graph edges to generate a set of "speed tiles". These speed tiles can be thought of as "lookaside" speed tables which are used by Valhalla dynamic costing methods to produce traffic-influenced routes. The proof of concept shows how route paths change and estimated times for the route increase in the presence of congestion.

Speed Tiles

A key method used in the proof of concept is to create speed tiles that correlate to the existing Valhalla graph edges. The speed data is stored in a 1:1 correlation to each graph edge. This allows easy and efficient access of speed data using the existing Valhalla graph IDs that index each graph edge. Storing dynamic speed data separately from the static graph tile data also allows the dynamic speed information to be read, cached, and updated separately without impacting the more static routing tiles. (The dynamic speed data is much smaller than the static graph tiles and can readily be updated and read as new routes are created. Valhalla graph tiles can remain cached.)

Speeds can be represented using a single byte per graph edge. This allows speeds from 0 to 255 kph. A specific value (e.g., 0) is used to indicate that no real-time speed exists for the edge and that the speed must be read from the Valhalla graph tile, which maintains a speed for each

graph edge derived from the OSM max_speed tag or approximated based on the OSM highway tag.

The proof of concept only considered real-time speed information which meant only a single speed is maintained for each edge. Historical speed data could also be supported as a set of speeds for specific time periods for each graph edge. For example, 168 different speed values could be stored to indicate the average speed along a road segment for each hour of the week. Historical speed data would be more static - it would not be updated every several minutes but could be read in and cached just as the Valhalla graph tiles are. Historical speed data can be used to provide time-dependent speed information that shows expected traffic patterns like rush hour commuting patterns vs. mid-day weekend traffic patterns.

Associating Way Ids to Valhalla Edges

One possible means of specifying speed or traffic information is to associate a current speed to an OSM way. This provides an easy method of adding speed data to Valhalla. An association of way Ids to Valhalla graph Ids was created for the traffic proof of concept. This was stored as a simple CSV (comma separated values) file listing the OSM way Id and the Valhalla graph Ids of the directed edges and their direction (forward or backward) along the way. A simple process was created to read a CSV file of way Ids with a forward direction speed and a reverse direction speed along the way. This process associated the way Ids to Valhalla directed edges and stored the corresponding speeds in a real-time speed file for each Valhalla tile where edges had real-time speeds were specified. This Valhalla real-time speed tile simply stores an array of speeds in a one to one correlation to the directed edges in the tile. If a directed edge did not have any speed assigned (the majority of edges) then a value of 0 was used to indicate no speed exists. Using real-time speed tiles in this manner allows the real-time speed to be accessed using the same Valhalla graph Id as the directed edge.

For the proof of concept the process that assigns speed to OSM ways, and thus Valhalla graph edges, was executed prior to running the Valhalla server so that the real-time speeds were populated and usable by Valhalla routing. Caching of the real-time speed data was implemented only on a per route basis. This meant that each new route would load real-time speed tiles that it needed. This allowed the proof of concept to inject updated speed information and then re-run a route and see the impact of the new or additional real-time speeds. The proof of concept involved a manual entry of speed information, it was in no way automated or using real traffic data. The purpose was to prove a method of influencing Valhalla routes using auxiliary speed tiles. Methods to ingest traffic information and automatically create speed tiles was outside the scope of the proof of concept.

The downside of using OSM way IDs for traffic specification is that OSM ways can be very long or very short. This can lead to difficulties when trying to localize congestion. OSM ways are often defined in such a way that manual addition or editing of a road is simplified. They often span many intersections or a long stretch of highway. When this happens, a single speed will become assigned to many graph edges and there is no way to represent variability of speed along the way. Conversely, OSM ways can be very short and represent only a small portion of a road between two intersections or can represent a small overpass on a highway. In this case it becomes difficult to assign speeds to many of these short OSM ways, leading to gaps in speed coverage.

Dynamic Costing with Traffic

Valhalla uses dynamic, run-time costing when computing route paths. Costing methods often use speed and edge length to compute time as the costing parameter and thus create least-time routes. Currently the speed used in these computations comes from OSM max_speed tags or from a speed assigned based on highway tags (if no max_speed tag is present). A custom, dynamic costing method was created for the traffic proof of concept. This costing method uses real-time speeds if available and falls back to the OSM assigned speeds if not available.

The proof of concept did not consider how to handle edge transition costs. These are costs to traverse intersections and are used to approximate time spent stopped or waiting at intersections. With real-time traffic information many of these transition costs become part of the real-time speed for a segment of road that may traverse several intersections. It is likely that when good real-time speed coverage is available that the edge transitions costing will need to be updated to lessen the impact of transition costs and thus rely more on the real-time speed data.

Example

The following images show an example of traffic-influenced routing with Valhalla using the proof of concept described above. The examples below all use a former commuting route that I took from Elkridge, Maryland to the Applied Physics Laboratory near Laurel, Maryland.

The first image shows the route without any traffic influence. The main part of the path takes I-95 South to MD-32 West to US-29 South with an estimated time of 17 minutes.

Using the OSM way IDs along I-95 South, I created real-time speed tiles that set a current speed of 45 kph along I-95 South. With the dynamic costing method using the adjusted real-time speed data along I-95 the resulting new route detoured to take US-1 South to MD-32 West to US-29 South. US-1 has a lower speed limit and more intersections, so this route takes an estimated 19 minutes. This is shown in Figure 2.

Using the OSM way Ids along US-1 South, I added real-time speeds of 40 kph along US-1 in addition to the reduced speeds along I-95 South. With the updated real-time speed tiles and dynamic costing the new route follows local roads to MD-100 West to US-29 South. This route takes an estimated 22 minutes and is longer distance as well.

These examples show what you would expect as traffic worsens - the optimal route path may become longer distance and may take some lower class roads to find a detour. The time to reach the destination increases as the initial shortest time path becomes slower due to traffic.

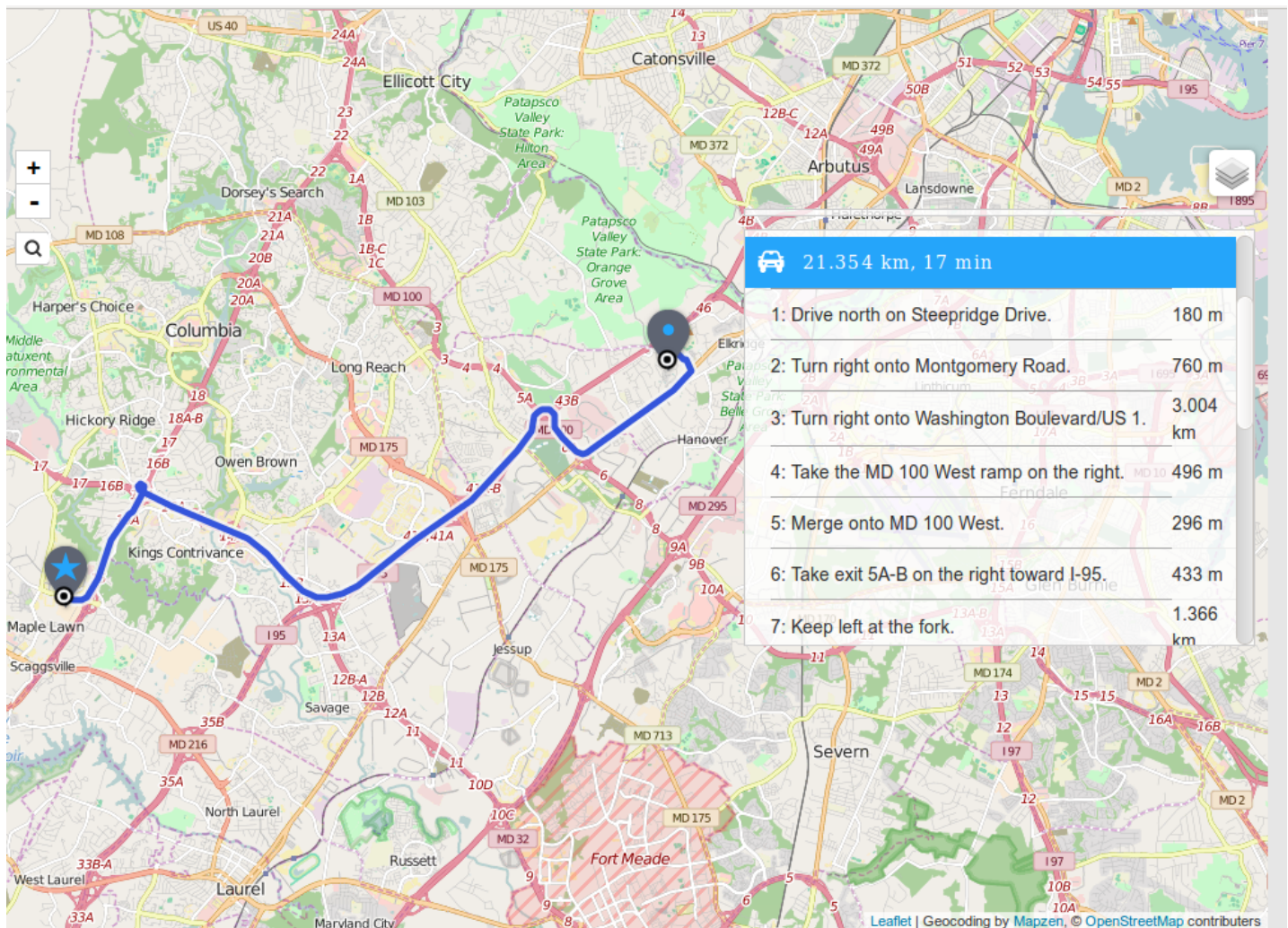
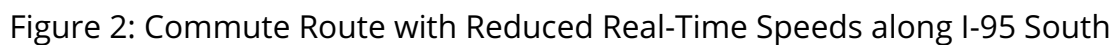


Figure 1; Commute Route without Real-Time Traffic







<https://mapzen.com/blog/speed-tiles/>

Valhalla's tiled data structures should work well for potential traffic integration. First and foremost, tiling allows for efficient distribution of regional sets of speed data and also allows distributed processing of traffic data based on tiles. The concept of "look-aside" speed tiles has several advantages:

- **Easy and rapid access** using the same indexes as the Valhalla graph edges.
- **Small data size for real-time speeds.** This is crucial since real-time speeds will need to be updated frequently to provide a robust and current routing solution in presence of traffic. Latency and delay in updating current speeds needs to be kept to a minimum in a responsive traffic-influenced routing system. Small data size for dynamic speed data also makes for more efficient access during route computation as less data needs to be read from disk and cached.

The one major disadvantage of the strategy of using look-aside speed tiles is that the speed tiles need to be matched to a specific Valhalla routing tile data set. Valhalla graph IDs are not persistent and depend on the data import process, so speed tiles need to be matched to a specific Valhalla data set so that indexes (graph IDs) match.

Work remains to be done to produce production-worthy, traffic-influenced routing within Valhalla. Here are some considerations and possibilities that we are considering.

Highway Hierarchies

Valhalla creates highway hierarchies in a manner similar to how roads are often presented at different zoom-levels in a map. The local hierarchy corresponds to the highest zoom levels where all roads and paths are stored or displayed. The arterial hierarchy removes residential roads, service roads, cycleways, walking paths, and other lower-class roads not generally used except when near the route origin or destination location. This is similar to a map at a middle zoom level. The highway hierarchy only includes motorways, trunks, and (currently) primary roads. This is similar to the lower zoom levels of a map where only important, higher classification roads are shown. In addition to the grouping of graph nodes and edges into the highway hierarchy, the arterial and highway hierarchy levels also contain "shortcut edges" that bypass any nodes that only connect to lower hierarchy level edges. This creates efficiency when computing long routes.

For the proof of concept, real-time speeds were only assigned to edges on the local graph hierarchy and thus were not assigned to shortcut edges. Since shortcut edges generally connect or combine edges with different OSM way IDs, the mapping of way IDs to Valhalla graph edges

becomes more complicated. We have left this mapping for future efforts if needed. Alternatively, a referencing system such as OpenLR might be used to associate traffic information to Valhalla routing edges. This has many advantages which we expect to investigate in the future.

Possible Optimizations Using Highway Hierarchies

We are considering a shift in the way Valhalla stores the routing graph in separate highway hierarchies. There is currently duplication of edges across different highway hierarchies. For example, a motorway edge is stored in all 3 hierarchies: local, arterial, and highway. This has some nice properties when used in a bidirectional, A* algorithm. In particular, the search paths only need to transition upwards in the hierarchy (from local to arterial to highway) and never needs to transition downwards. The 2 search paths meet in the middle, usually on the highway hierarchy for any driving route. Pedestrian and bicycle routes never transition upwards, they only traverse the local hierarchy.

If Valhalla were to store each edge only on the hierarchy level which it lies then this duplication would be removed. This reduces the total size of the Valhalla graph tiles and should lead to reduced memory use as well. For traffic integration this may have even more impact. If speed data is only provided for the arterial and highway hierarchies this would greatly reduce the number of traffic tiles required to support traffic and also would reduce the total size of any speed lookaside tiles. This seems like a reasonable assumption - residential roads, service roads, parking areas, and especially cycleways and walkways do not generally need traffic data as speeds are usually consistent and traffic volumes are usually too low to generate enough probe data to get meaningful traffic information.

This type of change to the Valhalla routing graph will need to be developed, validated, and tested to make sure routing performance and quality remains high. However, this idea seems promising and is worth pursuing as we move forward with traffic investigation and integration.

Keep watch for traffic integration work in the future!

· 18 July 2016 ·



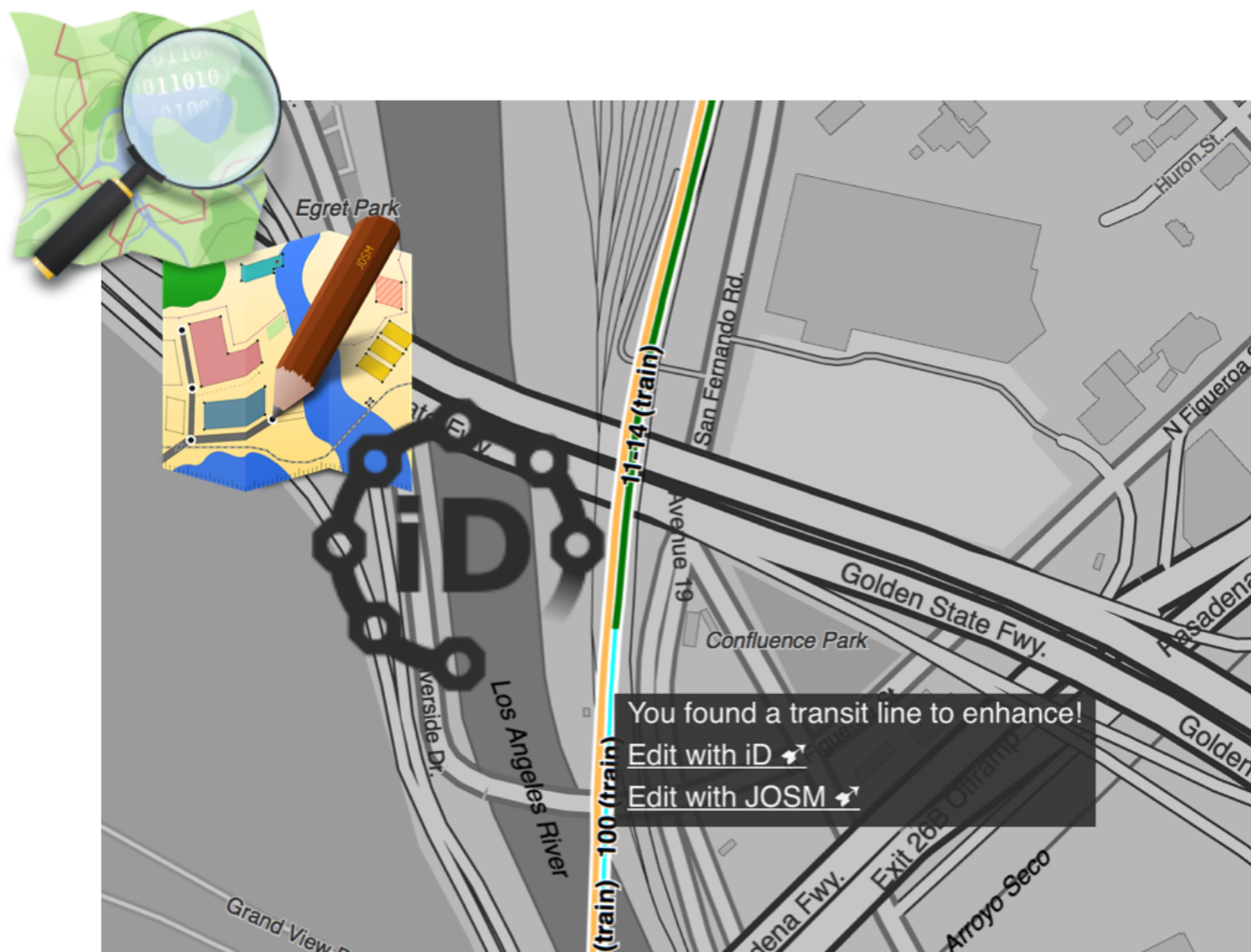
David Nesbitt

Dave leads Mapzen Mobility engineering. Rides a variety of 2 wheel vehicles.

© 2017 Mapzen

Targeted Editing Retrospective

osm (/tag/osm) targeted-editing (/tag/targeted-editing)



Icon Credits: **OpenStreetMap**

(https://en.wikipedia.org/wiki/OpenStreetMap#/media/File:Openstreetmap_logo.svg),

JOSM (http://wiki.openstreetmap.org/wiki/File:JOSM_Logo_2014.svg) & **iD**

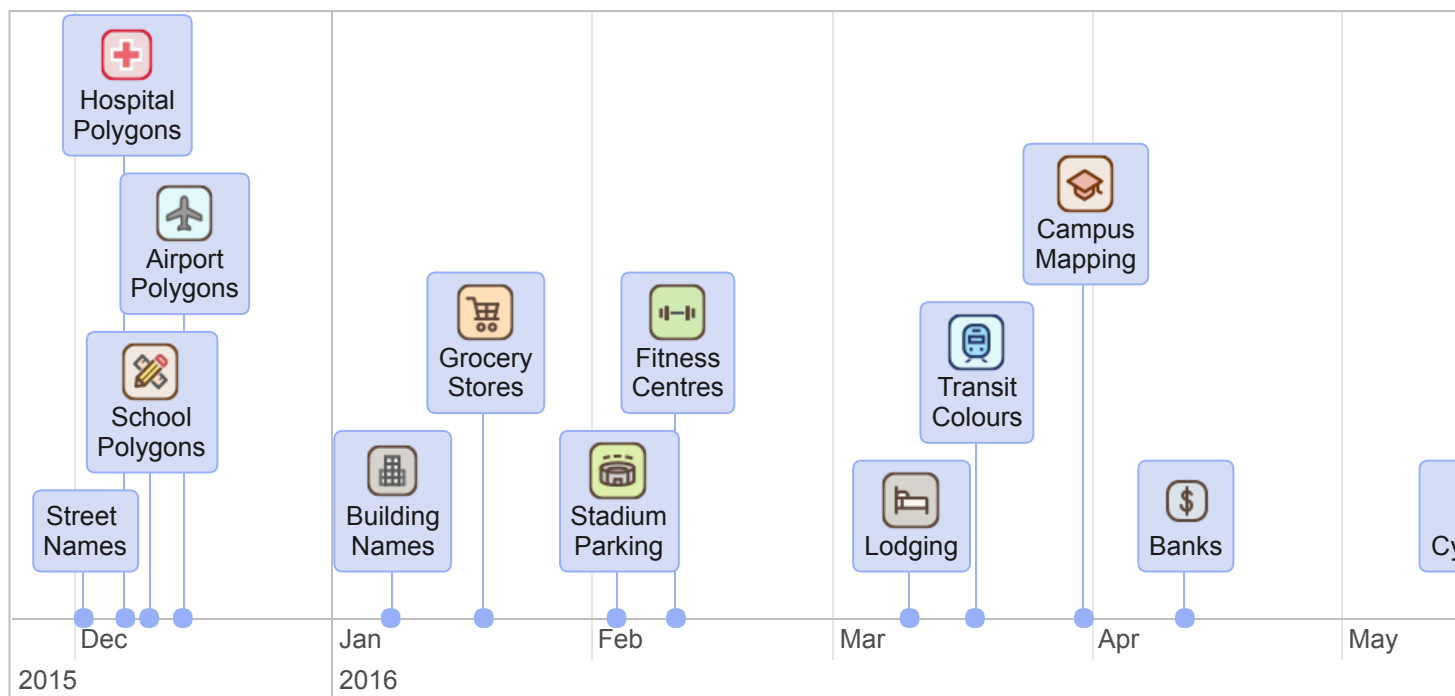
(<http://wiki.openstreetmap.org/wiki/File:ID.svg>)

Maps are current as of Oct 2016.

Targeted Editing, 8 months running. What have we learned?

Welcome to the **Targeted Editing** series where today you can watch cows grazing or make OpenStreetMap data amazing! (Both take about the same amount of time.)

We are nearly eight months in, can you believe it? How about a timeline?



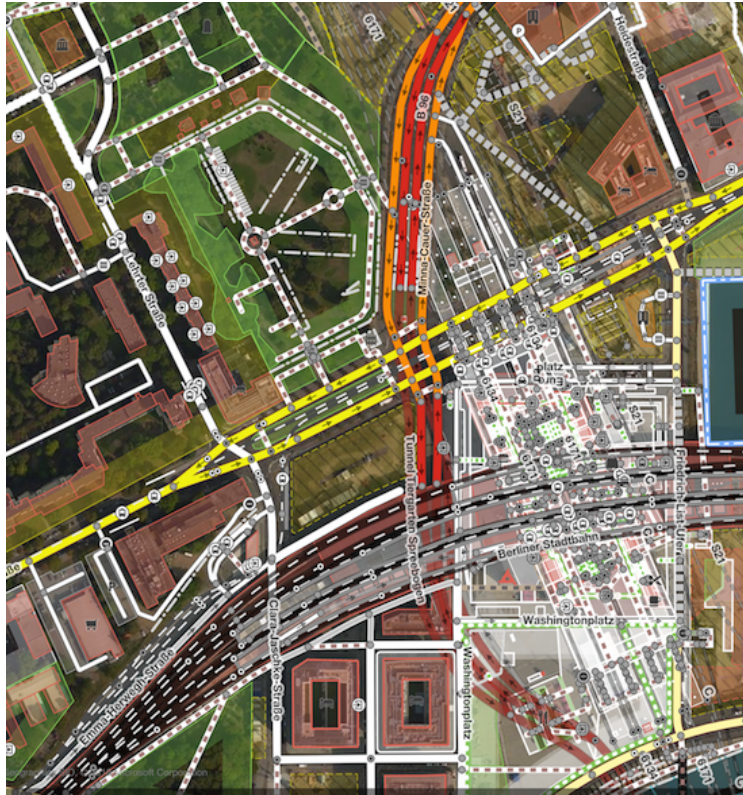
Icon Credits: **Mapzen Bubble Wrap** (<https://mapzen.com/blog/bubble-wrap-carto/>)

Since our first post in December 2015, there have been **13 posts** (<https://mapzen.com/tag/targeted-editing/>). Our most recent post was published in May of this year. It's a great time to take you behind the scenes of the Targeted Editing series to see how we come up with our post topics, how we identify which tags to query, how many edits have occurred, and how it all gets built into a map that you can use to quickly see where your help is needed.

Get your motor running with some motivation

At the core of OpenStreetMap's strength is it's community of editors. Thousands of editors are contributing local knowledge all over the world ever day to enhance the best open spatial database available. More and more editors are participating in remote mapping activities as well, often connected to well organized **Humanitarian OpenStreetMap** (<http://tasks.hotosm.org>) campaigns.

It is well understood that the spatial distribution of editors is not evenly distributed across the globe. In addition, some local communities of editors have been editing for significantly longer than other local communities. Ultimately, you will often find the greatest number of new editors joining from areas that appear to be extensively mapped. When faced with a screen full of map features, what can you add?



*Image Source: **OpenStreetMap** (<http://www.openstreetmap.org/#map=16/52.5253/13.3676>)*

Turns out, there are lots of features to add and existing features to enhance, even in the most densely mapped areas. Sometimes we all just need a little help finding things that can have a big impact as a result of a quick and simple edit.

A veritable theme park of map themes

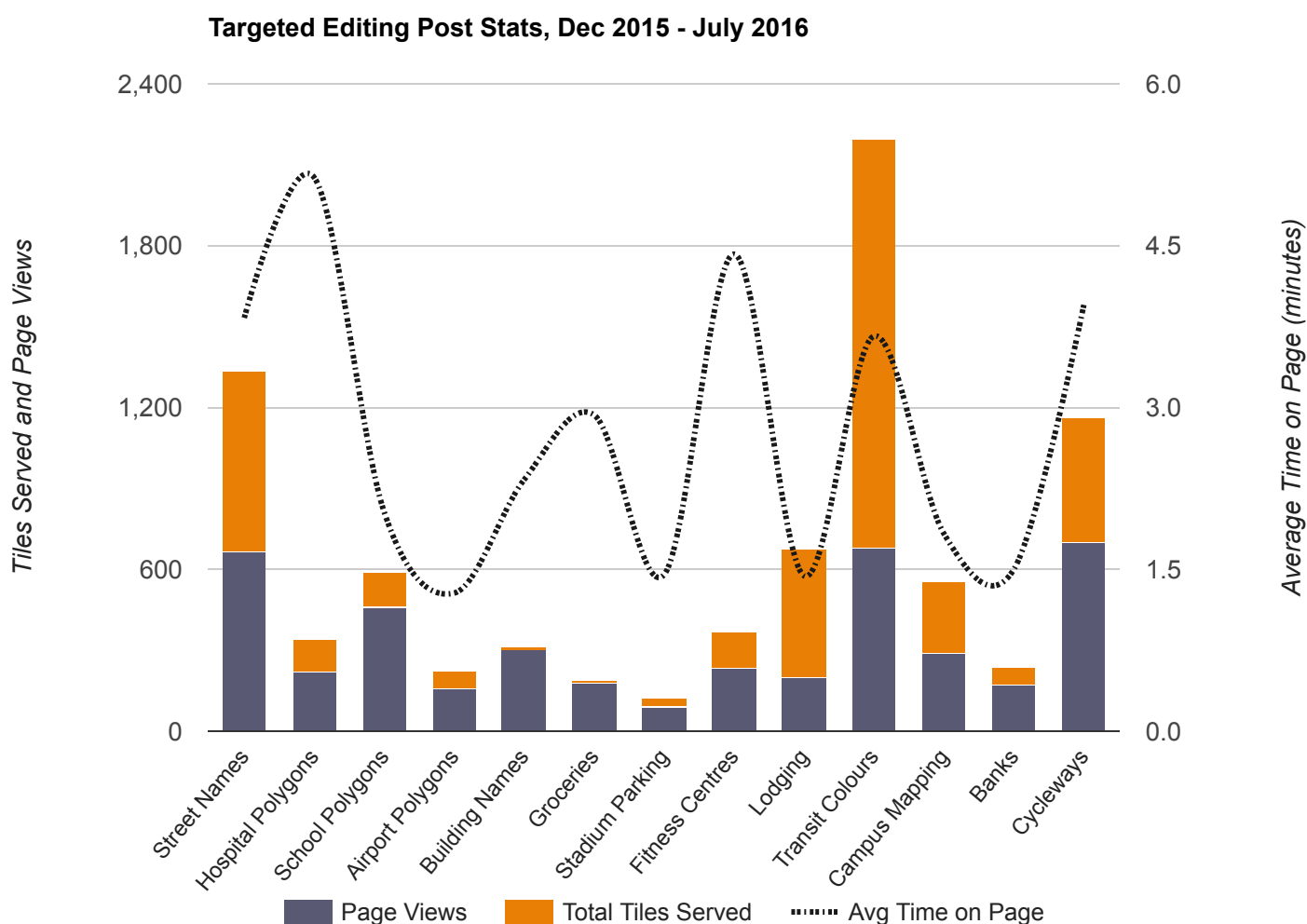
Every feature in OpenStreetMap has at least one tag to describe what it is. Care to guess how many distinct tags there are in OpenStreetMap? Go ahead, take a stab. I promise I will provide an estimate in a moment, but for now I will just say that it is a **very** large number.

Navigating this space is like visiting a theme park: lots of fun, sometimes dizzying, and often exhausting by the end of the day. To group themes of features into meaningful and manageable buckets, it is helpful to associate them with use cases. Of course the main use case is making a map for display, but it is common to build functionality on top of that. The two main examples are search and routing.

In the next few sections, we will break down our posts in relationship to these use cases. Oh! Are you still thinking about the number of distinct tags in OpenStreetMap? As of July 21st, there are 80,952,453 distinct tags! (*courtesy of taginfo* (https://taginfo.openstreetmap.org/reports/database_statistics))

Targeted Editing post engagement

To date, we have thirteen posts in the series. The school polygons and campus mapping posts have a little overlap. Ordered by their post date from left to right, the chart below shows a few variables that help us understand how engaged our readers and editors have been with each post.



The total page views for each post is shown in dark purple. Keep in mind: some posts have been up much longer than others. At the time of this writing, our first post was 229 days ago, and our most recent post was 60 days ago. Don't worry, new posts are in development.

One thing really stands out:

Posts involving highway tags received the most page views! This includes our **streets without names** (<https://mapzen.com/blog/targeted-editing-no-name-roads/>), **transit colours** (<https://mapzen.com/blog/targeted-editing-transit-colours/>), and **cycleway** (<https://mapzen.com/blog/targeted-editing-cycleways/>) posts. Perhaps this is no big surprise. There has always been a huge community around street edits, and it is exciting to see the community rallied around edits that contribute to routing applications.

The Targeted Editing series deployed a sub series at the beginning of the year to encourage edits to various types of businesses. Those posts, grouped as New Year's Resolutions, saw about a third of the traffic as compared to the highway tag related posts.

The post with the least amount of traffic goes to our **Tailgate Mania** (<https://mapzen.com/blog/targeted-editing-tailgate-mania/>) post designed to encourage stadium parking lot and parking aisle edits. While these types of edits stand to help routing, too, this post just did not garner much readership or edits.

Since each post is accompanied by a map to help editors find a theme of features to enhance, we are able to calculate the number of map tiles we have served for each post. The tiles served, divided across all days since the post went live, is shown in orange. These numbers come from our API keys. With the exception of school polygons and campus mapping, each post had it's own unique map and API key.

Our last measure is average time spent on page, reported in minutes. Overall, the numbers are good with a few surprises. Comparable with page views, the posts involving highway tags had longer average time on page, and the business listing posts were on the other end of the spectrum with the exception of the fitness centres and hospital polygon posts clocking in just over and just under 5 minutes each! The relationship between time on page and word count is muddy at best so we've left it out of our graph. The cycleways post was the longest at 3,040 words. The shortest was the building names post at 716 words.

Routing

Let's revisit the maps that were made to encourage routing related OpenStreetMap edits.

- [Edit Street Names](#)
- [Edit Bike Infrastructure](#)
- [Edit Parking Aisles & Stadiums](#)
- [Edit Transit Colours](#)

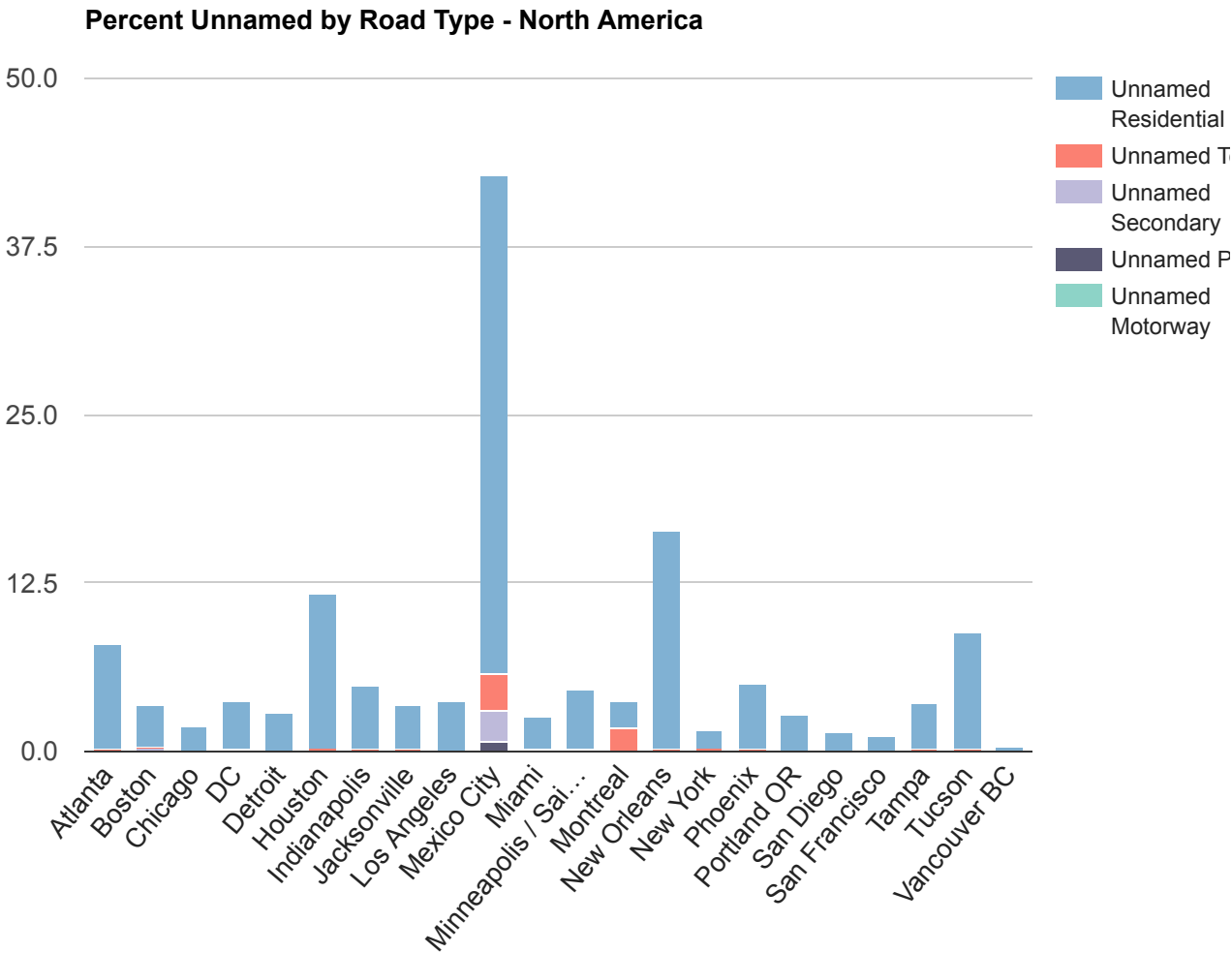
emojicons curtesy of [Emoji One](#)

Follow this link to interact with these maps full screen: **Routing Map Picker (https://mapzen-data.github.io/targeted-editing/retrospective/Routing_MapPicker.html)**. When viewing full screen, click the globe in the top right corner of the routing map picker any time to launch the active map tab full screen to search for other cities. Map some features! It's fun!

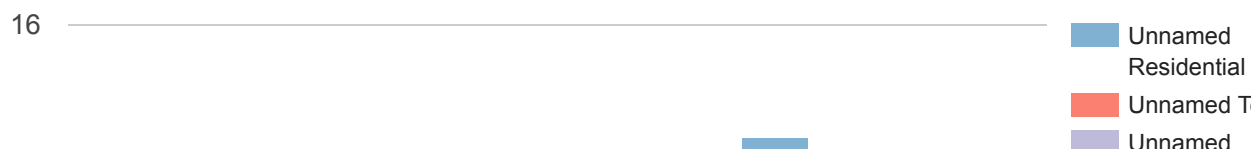
Our very first post featured **streets without names (<https://mapzen.com/blog/targeted-editing-no-name-roads/>)** and it still gets traffic after all this time. At first glance, the numbers can seem alarming. Are there really **that** many unnamed road segments? When you break things down, the more prominent motorways and primary roads are often well represented with names. The bulk of the unnamed road features are in the residential and tertiary categories. Overall, this is amplified in Mexico City and the subset of Asian cities presented below.

Unnamed Road Segments by Type

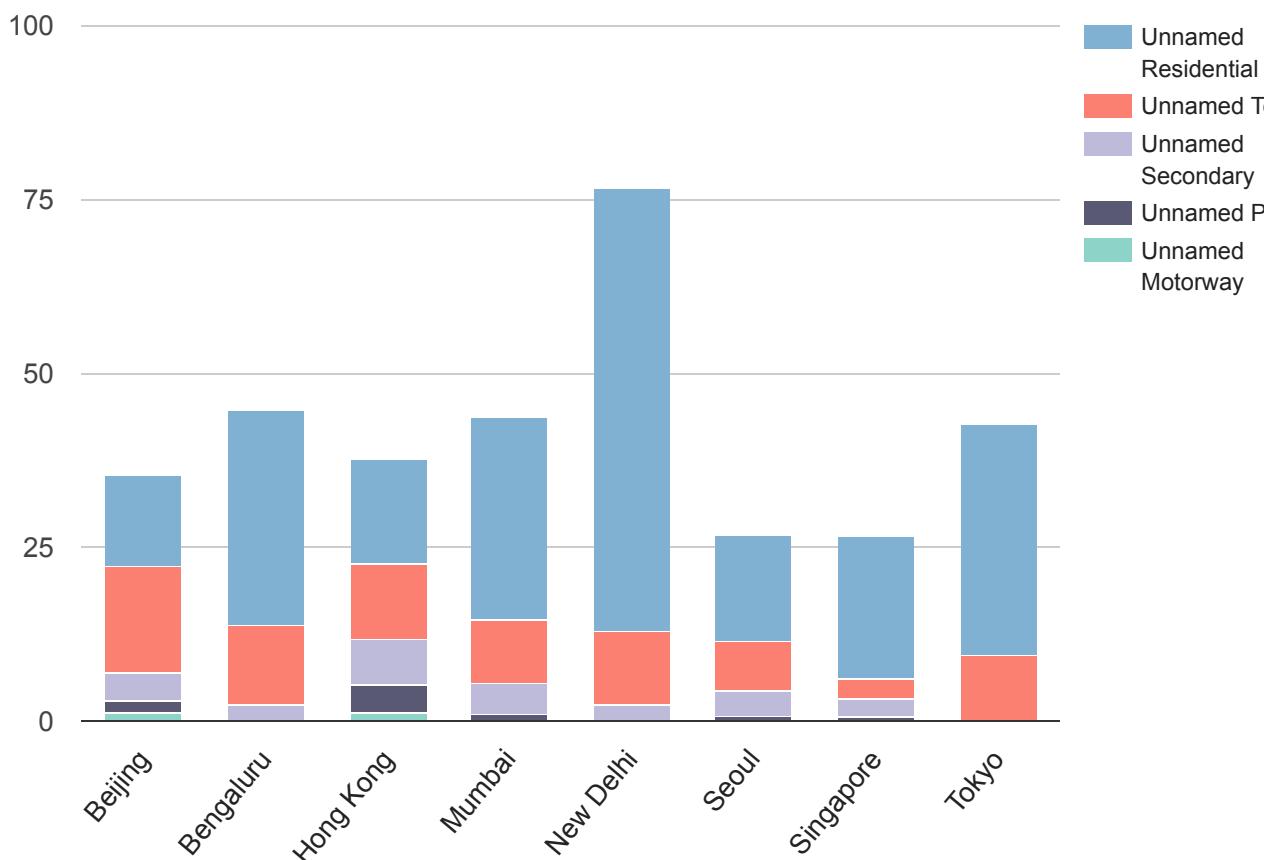
Click the legend items on the right on and off to isolate road types



Percent Unnamed by Road Type - Europe



Percent Unnamed by Road Type - Asia



Follow this link to interact with these charts full screen: **Unnamed Roads** (https://mapzen-data.github.io/targeted-editing/retrospective/Stacked_UnnamedRoadPicker.html)

In addition to unnamed street segments, the **cycleways** (<https://mapzen.com/blog/targeted-editing-cycleways/>) post got a lot of action. Across a subset of 45 metro extracts, 2,252 bike points of interest have been added or modified since the cycleways post, 39,633 kilometers of bike routes were added or modified, and 4,701 kilometers of bike tracks were added or modified! These values reflect an analysis of **Metro Extract** (<https://mapzen.com/data/metro->

extracts/) data made available on July 16th. We can't nail down the percentage of those edits that were a direct result of our cycleways post, but we are happy we have been able to participate in the push to inspire edits! These numbers alone help us understand how active the community is. One can't help but be inspired.

Our **transit colours** (<https://mapzen.com/blog/targeted-editing-transit-colours/>) post lies at the intersection of routing and display. The goal was to encourage editors to add colour to transit lines in OpenStreetMap where appropriate, and potentially boost the addition of some missing transit lines. Looking back on the twelve cities featured in the original **transit colours** (<https://mapzen.com/blog/targeted-editing-transit-colours/>) post, a lot has changed!

Percent of Route Relations with Colour

	Light Rail	Subway	Train	Tram
Beijing		77%		
Berlin	100%	100%	48%	81%
Boston	100%	100%	100%	100%
DC	100%	100%	100%	100%
London		100%	33%	100%
Moscow		91%	2%	
New York	84%	99%	99%	
Paris	100%	100%	95%	96%
San Francisco	100%	100%	100%	100%
São Paulo		100%	100%	
Sydney	100%		50%	
Tokyo	19%	99%	9%	

As of July 16th, 2016. For prior values, see the **March transit colours** (<https://mapzen.com/blog/targeted-editing-transit-colours/>) post.

The changes are definitely a sign of an active community of editors. Digging deeper, some features were reclassified, some were extended, and some were removed. Transit is dynamic.

Display

All of our posts encourage edits that support a number of use cases, but turning points into polygons is an excellent boost for display. There are many **airports** (<https://mapzen.com/blog/targeted-editing-airport-polygons/>), **hospitals** (<https://mapzen.com/blog/targeted-editing-hospital-polygons/>), and **schools** (<https://mapzen.com/blog/targeted-editing-school-polygons/>) in OpenStreetMap that are only represented as points. Some originate from early imports.

Digitizing a polygon to encompass the grounds for these features does wonders for their visual prominence on the map. The area of a feature can also be taken into consideration when calculating min zoom levels and search return priority. Area is not a perfect metric for these purposes, but it is a helpful metric. Check out the maps below from these posts:

- [Edit Airport Polygons](#)
- [Edit Hospital Polygons](#)
- [Edit School Polygons](#)

emojicons curtesy of [Emoji One](#)

Follow this link to interact with these maps full screen: **Display Map Picker** ([<https://mapzen.com/blog/targeted-editing-retrospective/>](https://mapzen-</p></div><div data-bbox=)

data.github.io/targeted-editing/retrospective/Display_MapPicker.html). When viewing full screen, click the globe in the top right corner of the Display Map Picker any time to launch the active map tab full screen to search for other cities.

Read on for graphs of new and modified points of interest since these posts were first published.

Search

What do you search for when you use maps? Common examples include addresses, businesses, and points of interest. We have featured Target Editing posts on **banks** (<https://mapzen.com/blog/new-years-resolutions-money/>), **building names** (<https://mapzen.com/blog/targeted-editing-name-that-building/>), **fitness centres** (<https://mapzen.com/blog/new-years-resolutions-fitness/>), **grocery stores** (<https://mapzen.com/blog/new-years-resolutions-groceries/>), and **lodging** (<https://mapzen.com/blog/new-years-resolutions-travel/>). Airports, hospitals and schools were mentioned above.

- [Edit Banks](#)
- [Edit Building Names](#)
- [Edit Fitness Centres](#)
- [Edit Grocery Stores](#)
- [Edit Lodging](#)

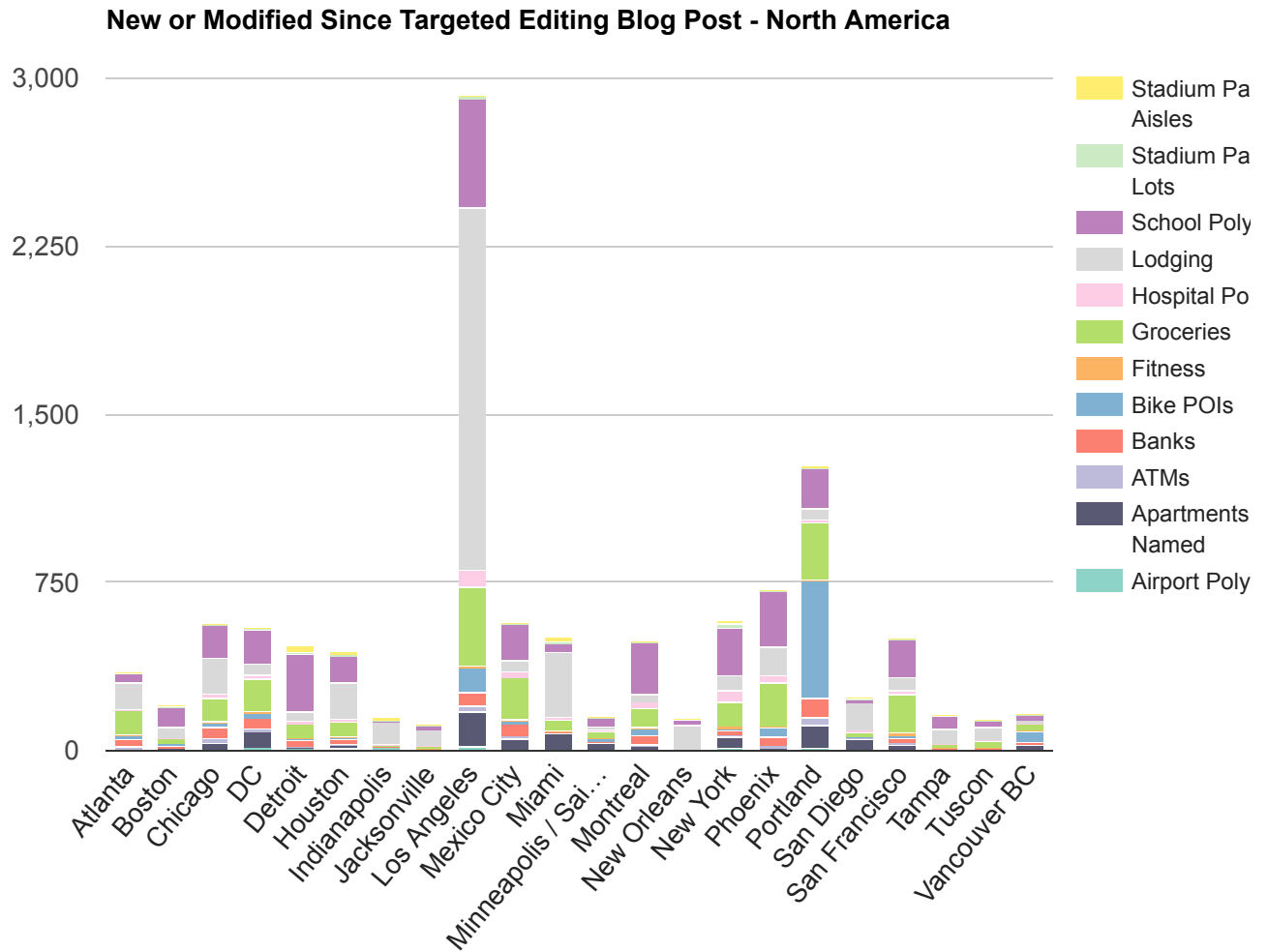
emojicons curtesy of [Emoji One](#)

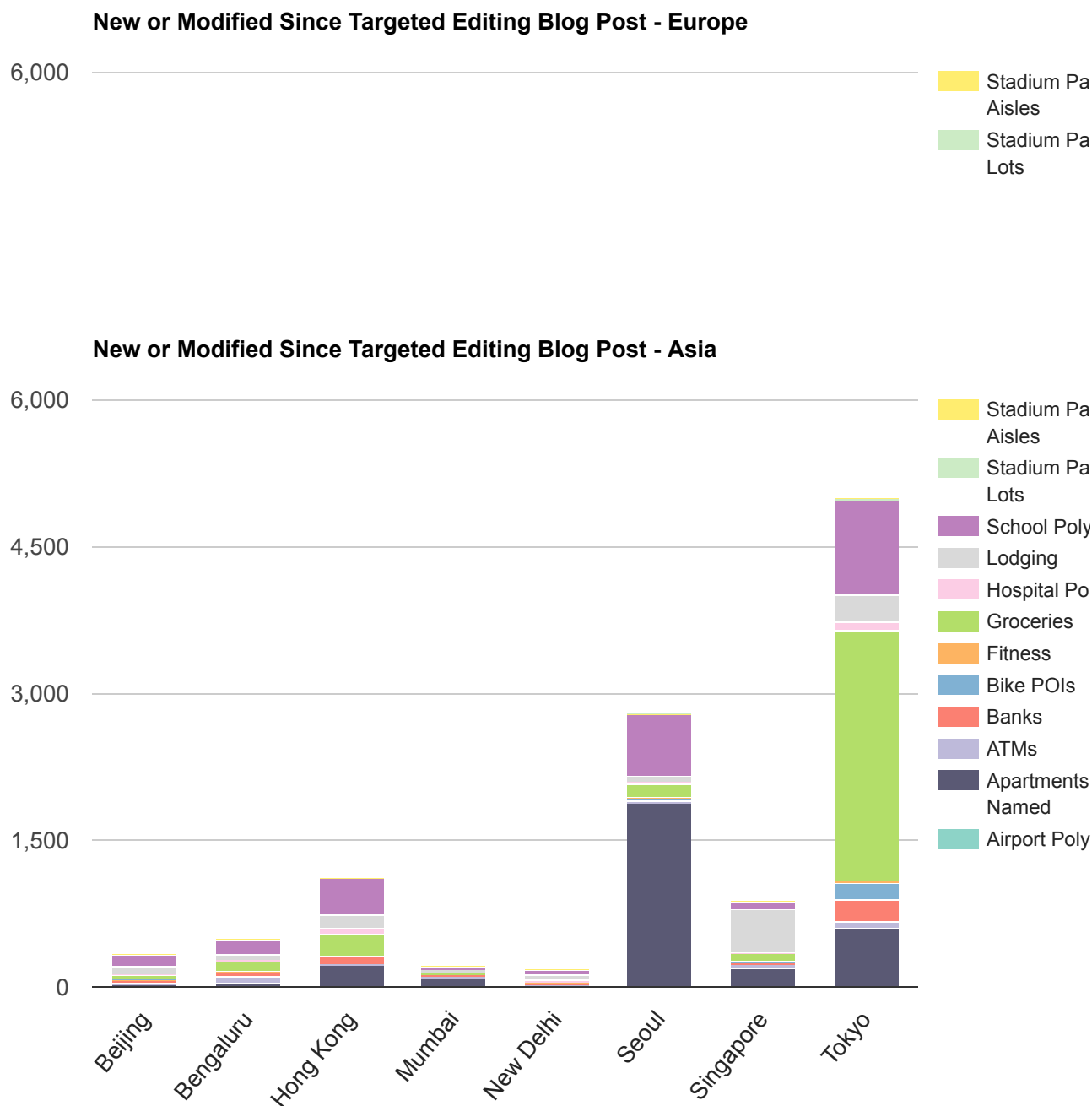
Again, these values reflect an analysis of **Metro Extract** (<https://mapzen.com/data/metro-extracts/>) data made available on July 16th. Follow this link to interact with these maps full screen: **Search Map Picker** (https://mapzen-data.github.io/targeted-editing/retrospective/Search_MapPicker.html). When viewing full screen, click the globe in the top right corner of the Search Map Picker any time to launch the active map tab full screen to search for other cities.

The world is filled with so many search query opportunities, it's easy to build Targeted Editing posts around them. The question is, are OpenStreetMap editors interested in adding them?

Points of Interest

Click the legend items on the right on and off to isolate feature types





Follow this link to interact with these charts full screen **Points of Interest** (https://mapzen-data.github.io/targeted-editing/retrospective/Stacked_PoiPicker.html)

School and hospital edits in Europe have had strong support from the Quarterly Projects for the **UK** (http://wiki.openstreetmap.org/wiki/UK_Quarterly_Projects) and **Belgium** (http://wiki.openstreetmap.org/wiki/WikiProject_Belgium/BE_Quarterly_Projects) along

with **International Women's Day** (<https://openstreetmap.us/2016/02/womens-mapathon/>) mapathons held in February. Schools in general receive more attention in our subset of North American and Asian cities, too.

Our New Year's Resolution posts featuring business points of interest tend to highlight lodging and grocery stores as more popular features to edit when compared to banks, ATMs and fitness centres. It's very possible that some imports are at work to generate these numbers. Nearly 2,500 grocery store edits in Tokyo and another 2,300 in Moscow stand out. A spike in lodging features in Los Angeles is a result of the ongoing **Los Angeles County building import** (<https://github.com/osmlab/labuildings>). Very exciting to see those buildings coming in with classifications and heights!

Building the Maps

The journey begins with the data

Data, data everywhere - how does it get on the map? After a post topic is chosen, a little data deep diving takes place. All of our topics have had well established **OpenStreetMap wiki pages** (http://wiki.openstreetmap.org/wiki/Map_Features) to help us identify the recommended and proper way to tag things. With this information, the next step is to pay **taginfo** (<https://taginfo.openstreetmap.org>) a visit to see what other tagging realities lie waiting for us in the data. Here is a walk through for our **New Year's Resolution - Travel** (<https://mapzen.com/blog/new-years-resolutions-travel/>) post:

1. What's the tag for lodging? For the entire grounds, the wiki suggests **tourism=hotel** (<http://wiki.openstreetmap.org/wiki/Tag:tourism%3Dhotel>) and for buildings: **building=hotel**. The wiki also lists a "See also" section with additional tags.
2. Are there any alternative, potentially less optimal, abandoned or unsanctioned tags for hotel? First thought: the **amenity** key, and **taginfo** (<http://taginfo.openstreetmap.org/search?q=amenity%3Dhotel>) confirms. For the blog post, we research the officially recommended ways for tagging features and present them, but for our maps and our queries, we include all variants. OpenStreetMap is a heterogeneous wonderland of variants. In this process, we find the use of the **leisure** key, too. Before you know it, you have a **giant query** (https://github.com/mapzen-data/targeted-editing/blob/gh-pages/queries/lodging_sql.sql#L4):

```
INSERT INTO lodging
    (query_name,
     value)
```

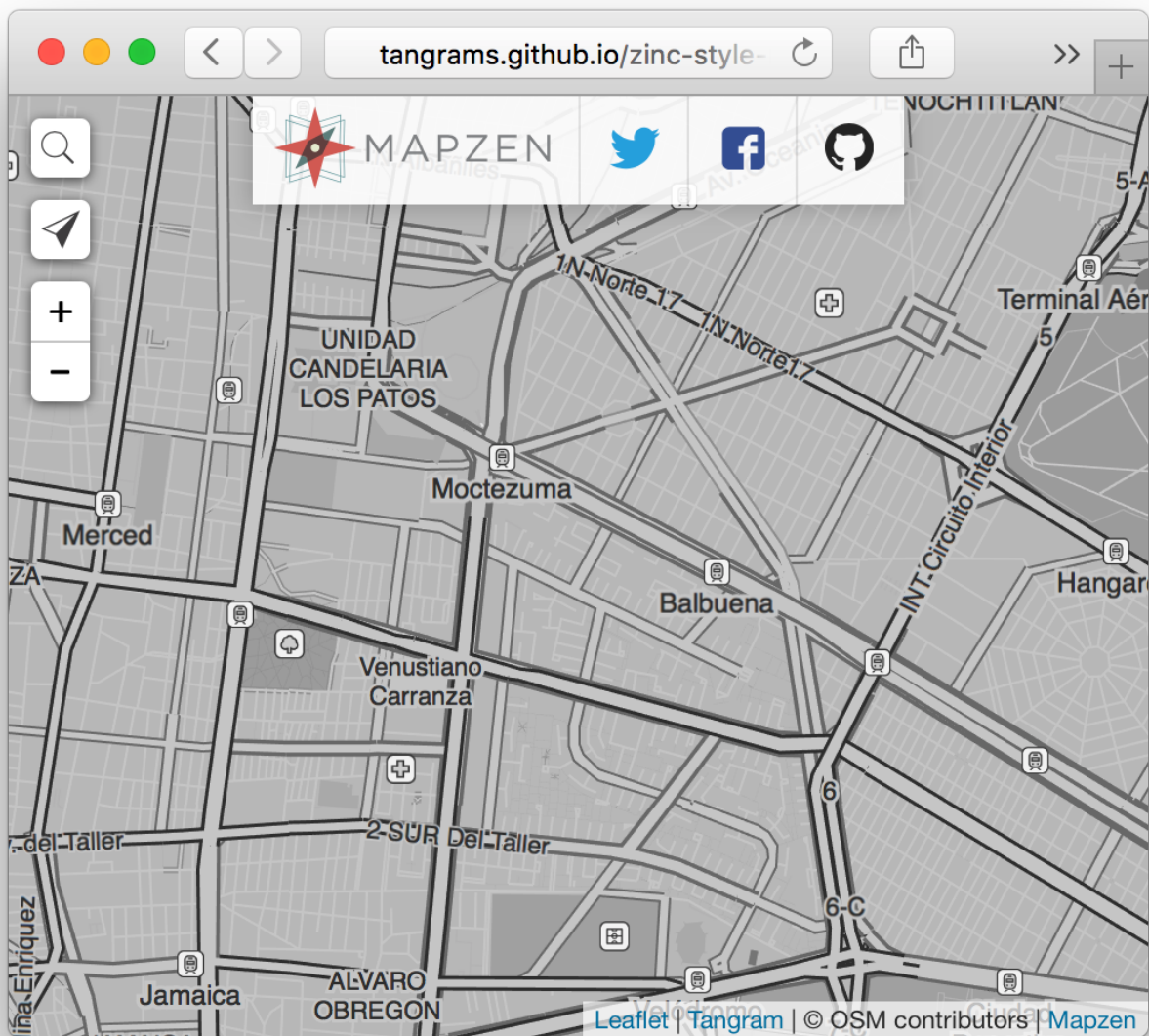
```
VALUES      ('lodging_points',
            (SELECT Count(*)
             FROM   planet_osm_point
             WHERE  ( tourism IN ( 'chalet', 'guest_house', 'hotel', 'hostel',
                                   'resort' )
                    OR building IN ( 'chalet', 'guest_house', 'hotel',
                                      'hostel', 'resort'
                                    )
                    OR amenity IN ( 'hotel', 'love_hotel', 'motel' )
                    OR leisure IN ( 'beach_resort', 'Beach_resort',
                                      'ski_resort' )
                    OR ( tags ?| array ['guest_house=bed_and_breakfast',
                                      'ski_resort'] )
                  )
            AND osm_id > 0));
```

...and this is repeated for **planet_osm_polygon** (https://github.com/mapzen-data/targeted-editing/blob/gh-pages/queries/lodging_sql.sql#L8). Hopefully this catches most of what we are looking for when we think of places to stay while traveling, the topic of our blog post.

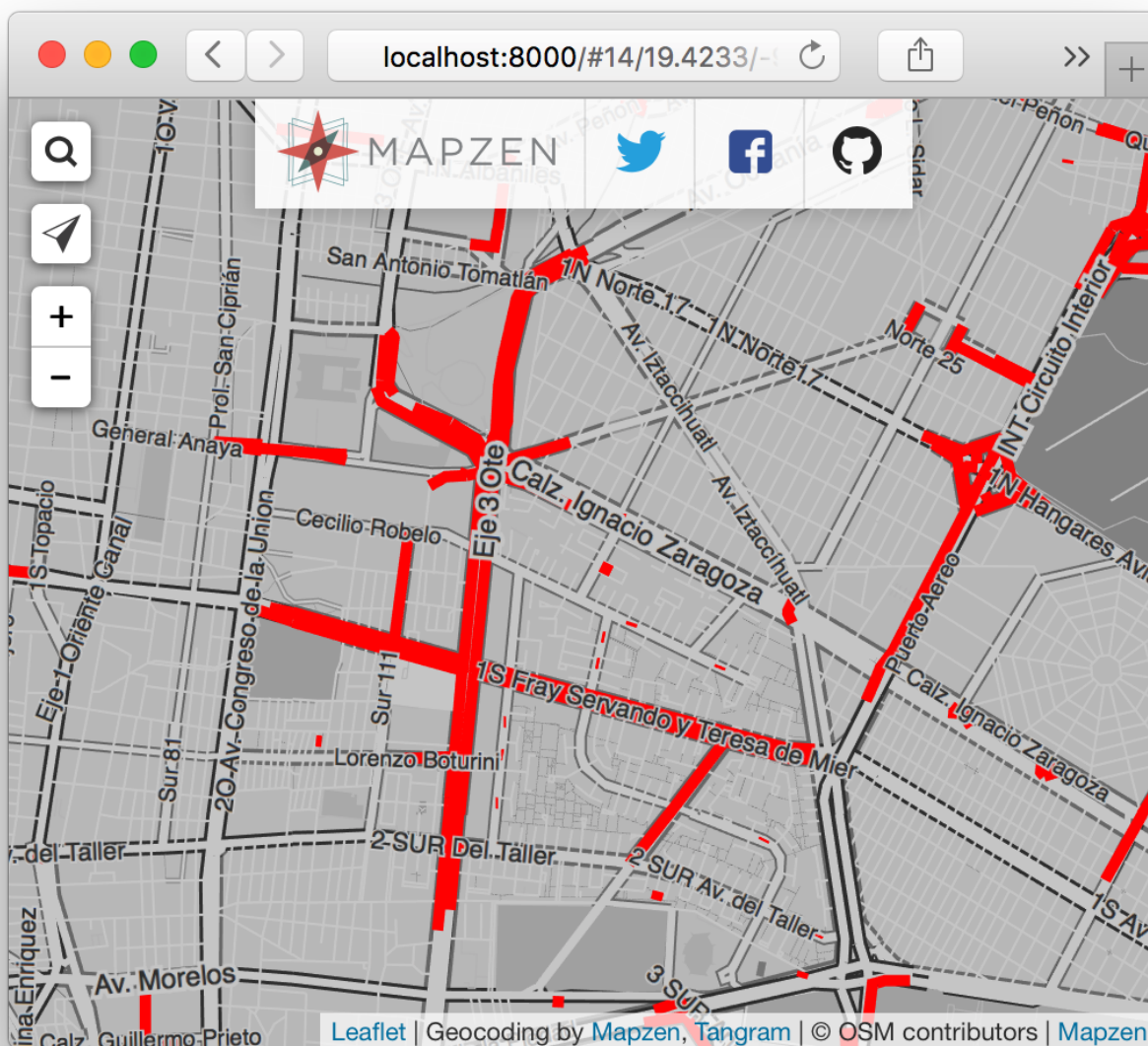
With a good start on queries, it is time to share all of this with Mapzen's Tilezen team to make sure our tiles actually contain most, if not all, of these features. **Mapzen's vector tiles** (<https://mapzen.com/projects/vector-tiles/>) consist of a **growing** (<https://github.com/tilezen/vector-datasource/blob/master/CHANGELOG.md>) subset of all OpenStreetMap features. For the New Year's Resolution series, the original plan was to start with the **fitness** (<https://mapzen.com/blog/new-years-resolutions-fitness/>) post, but those features needed to be added to our Mapzen tiles first, so we kicked off the series with the **groceries** (<https://mapzen.com/blog/new-years-resolutions-groceries/>) post.

Giving our target features visual prominence

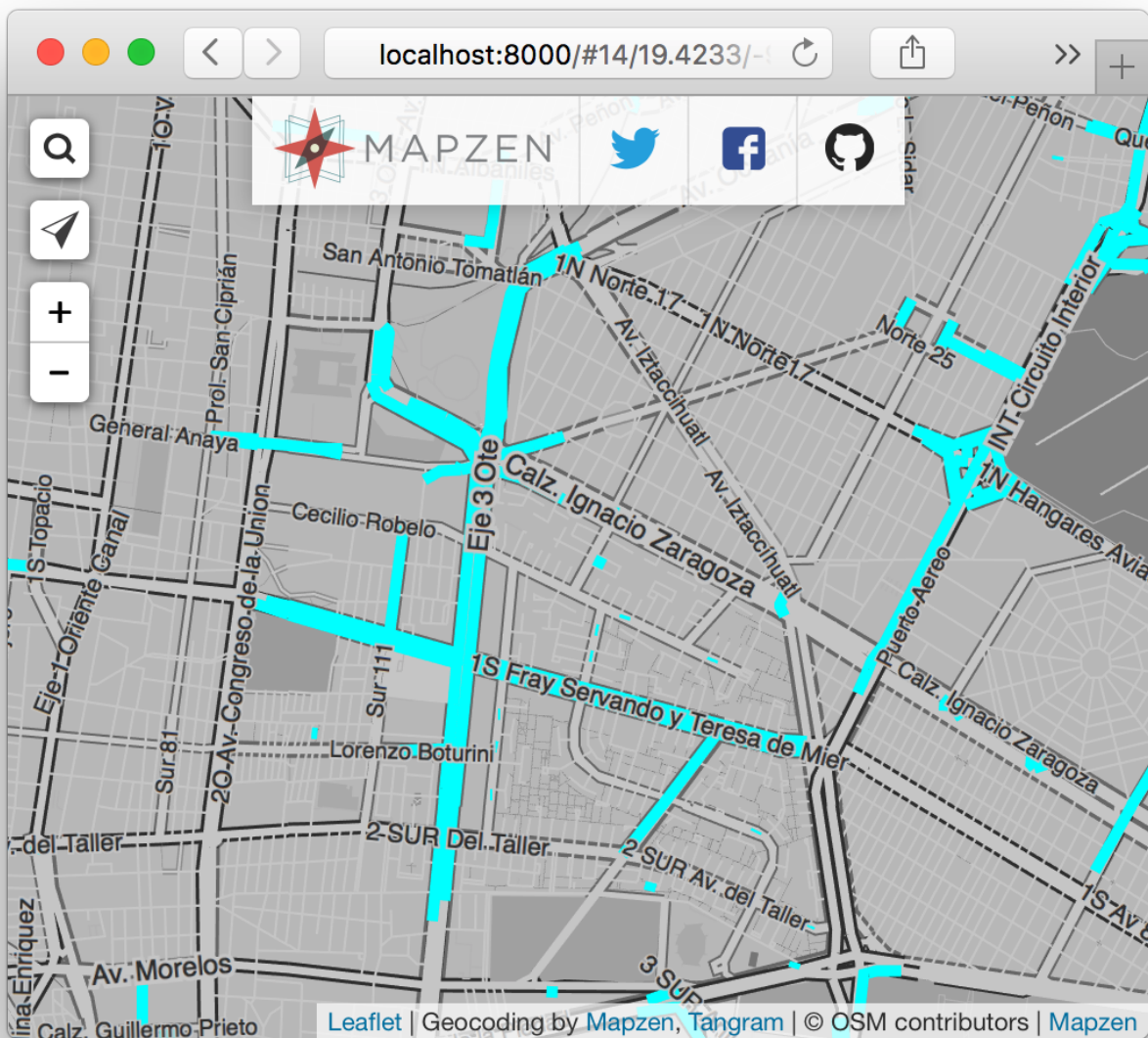
For our Target Editing series, we wanted to build an interactive map that highlighted features that might benefit from enhancements. Mapzen's **Refill and Zinc** (<https://mapzen.com/blog/introducing-refill-cinnabar-and-zinc-styles-for-tangram/>) map styles were both great candidates. Zinc won out in the end.



The color used to highlight features was originally red, and it really popped against the neutral grey shades of Zinc, but **red** sends an unintentional message: "Something is wrong."



Remember our original motivation: help editors find things to enhance. This could be adding names, street addresses, colour tags, and a number of other things. When a feature in OpenStreetMap is missing one of these tags, it does not necessarily mean that the feature is “wrong”. There actually are many roads that legitimately do not have names! So we ditched **red** and used **aqua** instead. Aqua, which is really similar to cyan, is a familiar highlight color that is bright and easy to see.



To create a custom version of the Zinc style, we can first import the original as a block and then write the extra lines of code to highlight our features of interest. A few extra considerations may be necessary, like ensuring that our features of interest draw on top of other features. Here's an example for the **streets without names** (<https://github.com/mapzen-data/targeted-editing/blob/gh-pages/te-street-names/map/roads.yaml#L673-L712>) featured above:


```

layers:
  nameless:
    data: { source: osm, layer: roads }
    filter: |
      function () {
        return (
          typeof feature["name"] === "undefined" &&
          typeof feature["ref"] === "undefined"
        );
      }
    matched:
      filter:
        highway: [ motorway, trunk, primary, secondary, residential, tertiary, road, living_street ]
        not:
          any:
            - kind: [path]
            - landuse_kind: [parking]
            - is_link: yes
            - aeroway: [runway, taxiway]
      draw:
        lines:
          interactive: true
          order: 1000
          color: aqua
          width: [[14, 8px], [16, 15]]

```

Building in map interactivity

With our custom style ready to go, the last step is pairing it with some JavaScript. The code results in a pop up window that appears when you **hover over a highlighted feature** (<https://github.com/mapzen-data/targeted-editing/blob/gh-pages/te-street-names/map/main.js#L78-L117>), and that window contains links to popular OpenStreetMap editing environments. A few features in this process were particularly important to us:

1. Clicking the link to **edit in iD or JOSM** (<https://github.com/mapzen-data/targeted-editing/blob/gh-pages/te-street-names/map/main.js#L97-L105>) should open these editing environments with the selected feature ready to go in edit mode. In densely edited areas, it can be difficult to click in just the right place to re-identify something we have identified for you.
2. Allow **shift+click** (<https://github.com/mapzen-data/targeted-editing/blob/gh-pages/te-street-names/map/main.js#L167-L181>) at any other location to open the iD editor centered over that location. This is particularly helpful for adding new features.
3. Allow **option+click** (<https://github.com/mapzen-data/targeted-editing/blob/gh-pages/te-street-names/map/main.js#L197-L205>) to display the topojson vector tile information.

With queries, yaml and JavaScript in place, we are ready to roll!

Addressing a limitation or two

With the Targeted Editing series maps, it's easy to highlight features that are missing a tag, and it's exciting to see the highlighting on a feature disappear when the feature is updated. Street names falls into this category.

What do you do when you want to highlight the absence of a related feature? The stadium parking and parking aisles falls into this category. In these cases, we highlighted the stadiums as reference points to help editors find them on the map. See a missing parking lot or missing parking aisles? Edit them in and connect them to the existing road network so routing engines can use them! Your stadium parking is updated, but the stadium still remains highlighted on the map. Not ideal, but it's a start in helping editors find features.

Blog post stats are static, meaning the stats calculated for the post are generated from the Metro Extracts that were available at the time. You get to enjoy the satisfaction of seeing the map tiles update with your handiwork, but the table of stats refuses to budge. Regular updates is a simple way to address this.

In summary

What have we learned? The Targeted Editing series has had fairly steady interest and engagement with our readers. Thank you! We are so glad you enjoy the posts.

Edits involving highway tags are by far the most popular of the topics we have tackled. Points of interest that fall into the business listing category are being added to OpenStreetMap, too, just not at the same rate as highway infrastructure. Perhaps the biggest take away is evident in the measures of activity over time. The growth of OpenStreetMap data is happening at a faster rate in North America and Europe, whereas India is not quite yet in the fast lane, but still chugging along. None of the cities we have looked at are at a standstill with respect to community edits and participation. Community effort is truly global.

And finally, we can't take credit for all of the edits that have taken place since each post was published, but we are more excited about that than anything else because we see lots of parallel initiatives to encourage edits. Open data is a remarkable game changer for communities, and we are really excited to be a part of it.

What would **you** like to see next in the series?

· 21 July 2016 ·

Indy Hurt



Indy was our resident data scientist lending her geographic expertise to all things "open".

© 2017 Mapzen

Seeking Mapzen at SOTMUS

osm (/tag/osm)

Are you heading to **Seattle** (<https://tangrams.github.io/bubble-wrap/#16.5165/47.60922/-122.31821>) for **State of the Map US** (<http://stateofthemap.us/>)? We'll have lots of folks there talking about geo things at the following sessions and lightning talks and workshops:

- **Beyond Aesthetic Icing: Designing geo tools for humans** (<http://stateofthemap.us/2016/beyond-aesthetic-icing/>) (Sat, 4:45) - *Ekta Daryanani, Meghan Hade*
- **Behind the Scenes of the Mapzen Targeted Editing Series - Engaging Editors** (<http://stateofthemap.us/2016/behind-the-scenes-of-the-mapzen-targeted-editing-series-engaging-editors/>) (Sun, 4:30) - *Indy Hurt*
- **Befriending a Geocoder** (<http://stateofthemap.us/2016/befriending-a-geocoder/>) (Sun, 11:45) - *Diana Shkolnikov*
- **Give your vector tile life** (<http://stateofthemap.us/2016/give-your-tile-life/>) (lightning talk, Sat, 4:45) - *Hanbyul Jo*
- **The Space Between: expanding street centerlines to street polygons** (<http://stateofthemap.us/2016/project-lead/>) (lightning talk, Sat, 4:45) - *Lou Huang*
- **Make maps more interactive with the magic of a geocoding search box** (<http://stateofthemap.us/workshops/>) (workshop, Mon 9AM) - *Rhonda Glennon, Katie Kowalsky, Diana Shkolnikov*

And stop by our table and say hi and ask questions! We'll also have a table full of geo swag and geo stickers and geo people!



And **Bubble Wrap** (<https://mapzen.com/blog/bubble-wrap-carto/>) POI stickers!



And more!

Come by and say hi!

· 22 July 2016 ·



Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.



John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

VPC Architecture – Bridging Regions

engineering (/tag/engineering)

Mapzen is expanding!

As Mapzen continues to grow – both in users of the various platform services and in the services we offer – geo distribution of those services becomes a necessity. While we strive to edge cache as much content as we can, there are various scenarios that necessitate origin servers in regions across the world.

While simply building new application servers in other regions is not an incredibly complicated task, it doesn't address a multitude of other issues that arise when building infrastructure in multiple AWS VPCs that have no private network interconnectivity. Namely, what do we do about monitoring infrastructure, cases where we have database masters in a single region that we need to talk to from all regions, and all the other edge cases that arise?

We remain cognizant of the fact that ideally, infrastructure in every region would be autonomous, but the reality is that there will always be situations that may require one of those regions to talk to another. One option is of course to tunnel those communications over the public internet. This would necessitate that those applications had an easy means of encrypting traffic, which isn't always the case. It would also mean opening up holes in security policies to allow the inbound communication, which carries with it a number of additional concerns. Rather than manage this sort of thing at the application level, we decided a global secure tunneling solution would be preferred.

This post will give a little taste of the theory and execution behind our setup. It'll be light on specific details, mainly because it'd be pretty tedious for me to write and you to read! But if you want to discuss the finer points of the implementation, **please get in touch** (<https://twitter.com/intent/tweet?text=@mapzen+bridging+VPCs>).

The Theory

There are a number of useful posts out on the web detailing, to various degrees, how to use **openswan** (<https://www.openswan.org/>) to connect AWS VPCs. After evaluating a few other software based solutions, we decided to stick with the general KISS principal and build a setup

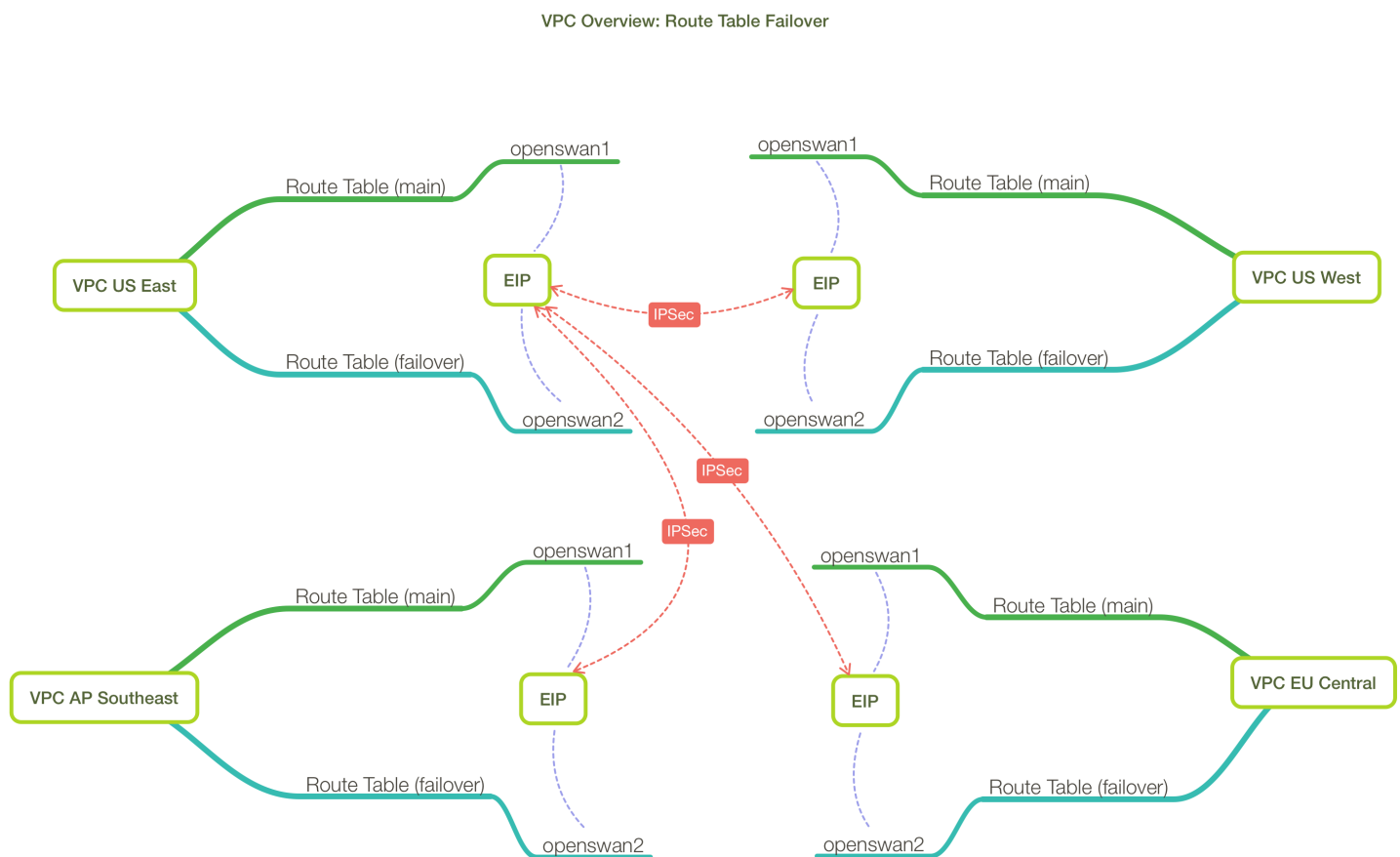
using openswan as well.

In addition, we elected to designate a 'primary' VPC, which could talk to all other regions we build and which all those regions could talk back to, but not to mesh all the other non-primary VPCs. Reducing the mesh reduces the complexity enormously, and we had no use case that would warrant doing it.

When building out the new VPCs, we also tried to maintain consistency in network addressing:

10.[even number]/16 subnets would be production, 10.[odd number]/16 subnets would be non-production. These small details become very important as your infrastructure continues to grow... don't discount them as insignificant!

How it looks



In each region, we have two systems running openswan. One is assigned an EIP at any given time and acts as the primary system through which traffic from another region or regions is flowing.

There are a number of, shall we say, interesting tweaks needed to these instances to allow all this to work. One example is MTU size, at least in our configuration where we're allowing traffic to flow from production to non-production VPCs over an AWS VPC peering:

```
*mangle
# Fix mtu craziness when tossing data around in ipsec tunnels
-A POSTROUTING -o eth0 -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --set-mss 1300
```

Each VPC has two route tables, a primary and a failover, as we're using static routing. So each route table knows to route traffic for each VPC through one of the openswan instances in its region. Obviously only one route table can be in use at any given time, which is where monitoring and failover come into play...

Monitoring and Failover

```
# get instances and eip from the stack
#
def get_instances(stack_id)
  client = Aws::OpsWorks::Client.new(region: 'us-east-1')
  resp = client.describe_instances(stack_id: stack_id)

  ...

# associate a specific route table with all the subnets
#   in a region, overriding the main route table.
#
# examples:
#   failover_rtb(rtb_failover, subnets, region, false)
#
def failover_rtb(routetable = '', subnets = [], region = '', dryrun = true)
  client = Aws::EC2::RouteTable.new(region: region, id: routetable)

  ...
```

We wrote a bit of monitoring infrastructure code (sampled above), in the form of a Sensus subscriber check which all these systems run. The check determines what region it's being run in and whether it's being run by the primary (EIP assigned) system or not. With that information, the systems then ping the other regions to ensure that connectivity always exists. In the event a failure threshold is reached, a failover is initiated: the EIP is swapped to the other openswan instance, and the matching route table for the VPC in question is enabled. In the interim, informational alerts are generated to let us know this has occurred, but guess what? We don't have to do anything to intercede! (And who doesn't love that kind of automation?)

Conclusion

So that's the thousand foot overview. As mentioned earlier, if you're interested in more details, **get in touch** (<https://twitter.com/intent/tweet?text=@mapzen+bridging+VPCs>), and conversely if you've implemented something similar we'd **love to learn more about your setup** (<mailto:hello@mapzen.com?subject=connecting%20AWS%20VPCs>). In the meantime, we're enjoying passing encrypted traffic between datacenters strewn across the globe as we continue to build and improve our infrastructure.

· 27 July 2016 ·



Grant Heffernan

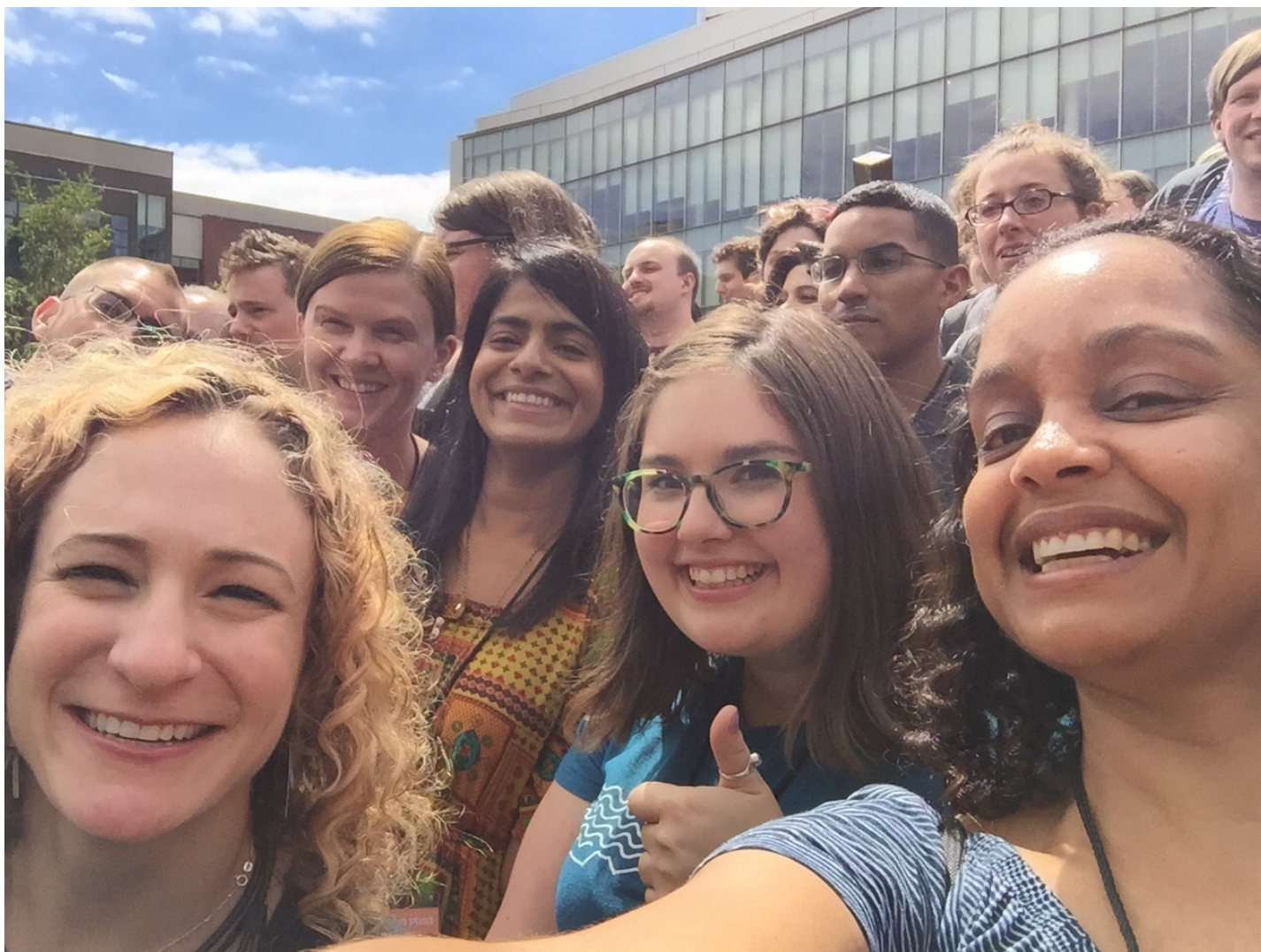
Grant is a sysadmin with fingers in all of Mapzen's pies. Egli non si senta italiano, ma per fortuna o purtroppo...

© 2017 Mapzen

Rolling up Seattle

osm (/tag/osm)

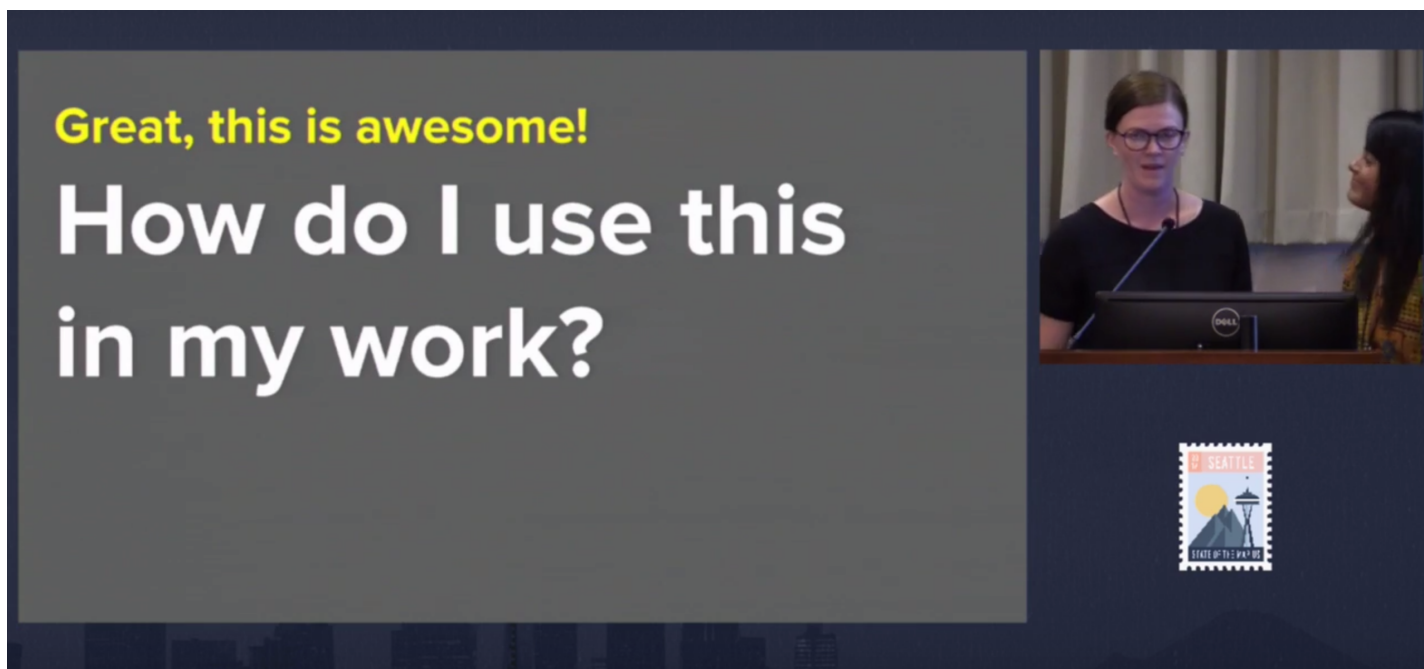
It was most excellent to see everyone in Seattle, whether it was in the halls, during the presentations, workshops, and lightning talks, or at our table!



Mapzen staff selfie during the SOTMUS group photo

If you weren't able to attend, or missed any of talks, we've listed them below. SOTMUS has kindly **uploaded all of the videos** (<https://www.youtube.com/playlist?list=PLqjPa29IMiE3eR-gK80irr3xdUiRbIMeg>). We've also gathered all of our **presentations and materials in one spot** (<https://github.com/mapzen/presentations/tree/master/07-2016-SOTMUS>).

Beyond Aesthetic Icing: Designing geo tools for humans




presentation by Ekta Daryanani, Meghan Hade – **video** (<http://stateofthemap.us/2016/beyond-aesthetic-icing/>) and **slides** (<https://github.com/mapzen/presentations/tree/master/07-2016-SOTMUS/BeyondAestheticIcing>)

Behind the Scenes of the Mapzen Targeted Editing Series – Engaging Editors

Aqua

presentation by Indy Hurt – video (<http://stateofthemap.us/2016/behind-the-scenes-of-the-mapzen-targeted-editing-series-engaging-editors/>) and materials (<https://mapzen.com/blog/targeted-editing-retrospective/>)

Befriending a Geocoder



hortly thereafter some of the finest universities in the US, such as Harvard and Yale, followed suit and implemented their own versions of early geospatial search engines. A team of Yale graduates and students developed a protocol they called the Dual Independent Map Encoding, DIME for short.

This groundbreaking protocol paved the way for geocoding algorithms still used in some of today's most popular commercial geocoders, such as Google and MapQuest.

<http://guides.library.yale.edu/GIS/geocoding>
<http://www.geog.ucsb.edu/~kclarke/G128/Lecture04.html>
https://en.wikipedia.org/wiki/Dual_Independent_Map_Encoding

DIME/GBF SYSTEM

Map 3

Block	Side	Block	Side	Block	Side	Block	Side
156	W	157	E	158	W	159	E
160	W	161	E	162	W	163	E
164	W	165	E	166	W	167	E
170	W	171	E	172	W	173	E
176	W	177	E	178	W	179	E
180	W	181	E	182	W	183	E
184	W	185	E	186	W	187	E
190	W	191	E	192	W	193	E
196	W	197	E	198	W	199	E
200	W	201	E	202	W	203	E

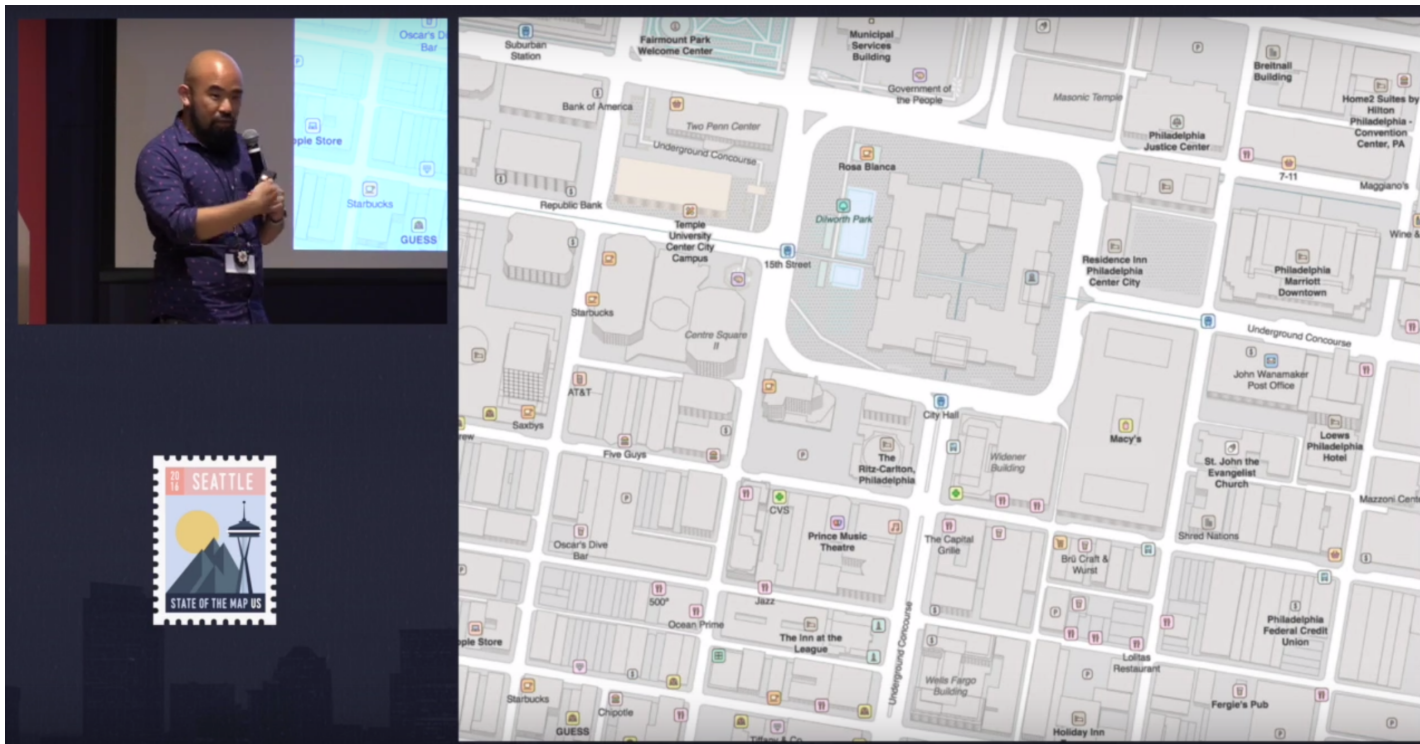
presentation by Diana Shkolnikov – video (<http://stateofthemap.us/2016/befriending-a-geocoder/>) and slides (<https://github.com/mapzen/presentations/tree/master/07-2016-SOTMUS/BefriendingGeocoder>)

Give your vector tile life



lightning Talk by Hanbyul Jo – video (<https://youtu.be/xOGsy9BFJ5Y?list=PLqjPa29IMiE3eR-gK80irr3xdUiRbIMeg&t=2195>) and notes (<https://github.com/mapzen/presentations/tree/master/07-2016-SOTMUS/give-your-vector-tile-life>)

The Space Between: expanding street centerlines to street polygons



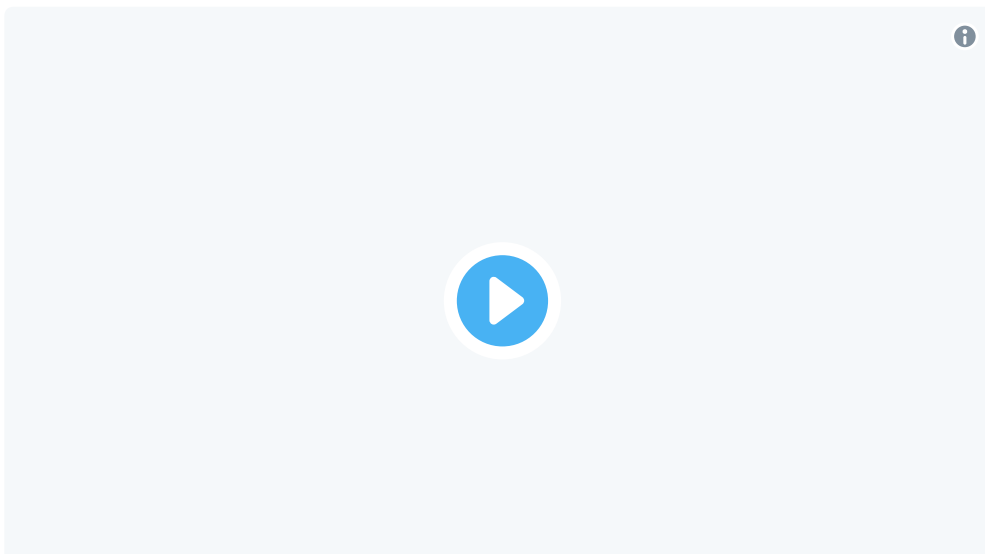
lightning talk by Lou Huang – **video** (<https://youtu.be/xOGsy9BFJ5Y?list=PLqjPa29IMiE3Er-gK80irr3xdUiRbIMeg&t=2523>) and **notes** (<https://github.com/mapzen/presentations/tree/master/07-2016-SOTMUS/The-Space-Between>)

Make maps more interactive with the magic of a geocoding search box



workshop by Rhonda Glennon, Katie Kowalsky, Diana Shkolnikov – **workshop materials**
(<https://github.com/mapzen/presentations/blob/master/07-2016-SOTMUS/geocoding-workshop/tutorial.md>)

And here's a video of the axidraw, drawing maps like olden times. We'll have a blog post on it soon.



**Michal Migurski**

@michalmigurski



Mesmerized watching the Axidraw rendering @Mapzen
OpenStreetMap data for Washington DC at #sotmus

12:44 PM - Jul 24, 2016

1 18 59

Many of you have been asking about our **Null Island t-shirts** (<https://nullis.land>). (Thanks to Brennan at **Workhorse Industries** (<http://www.workhorseind.com>) in Seattle for their fantastic printing work!)

**Mapzen** ✓

@mapzen



Introducing the new #NullIsland shirts at @sotmus!

9:52 AM - Jul 23, 2016

2 4 26

We ran out, but we've put the **vector artwork online** (<https://nullis.land>), and we're looking into on-demand printing! And come by a Mapzen event if you want Null Island, POI icon or Transitland stickers!

· 01 August 2016 ·

Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.



John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

Optimizing Your Route

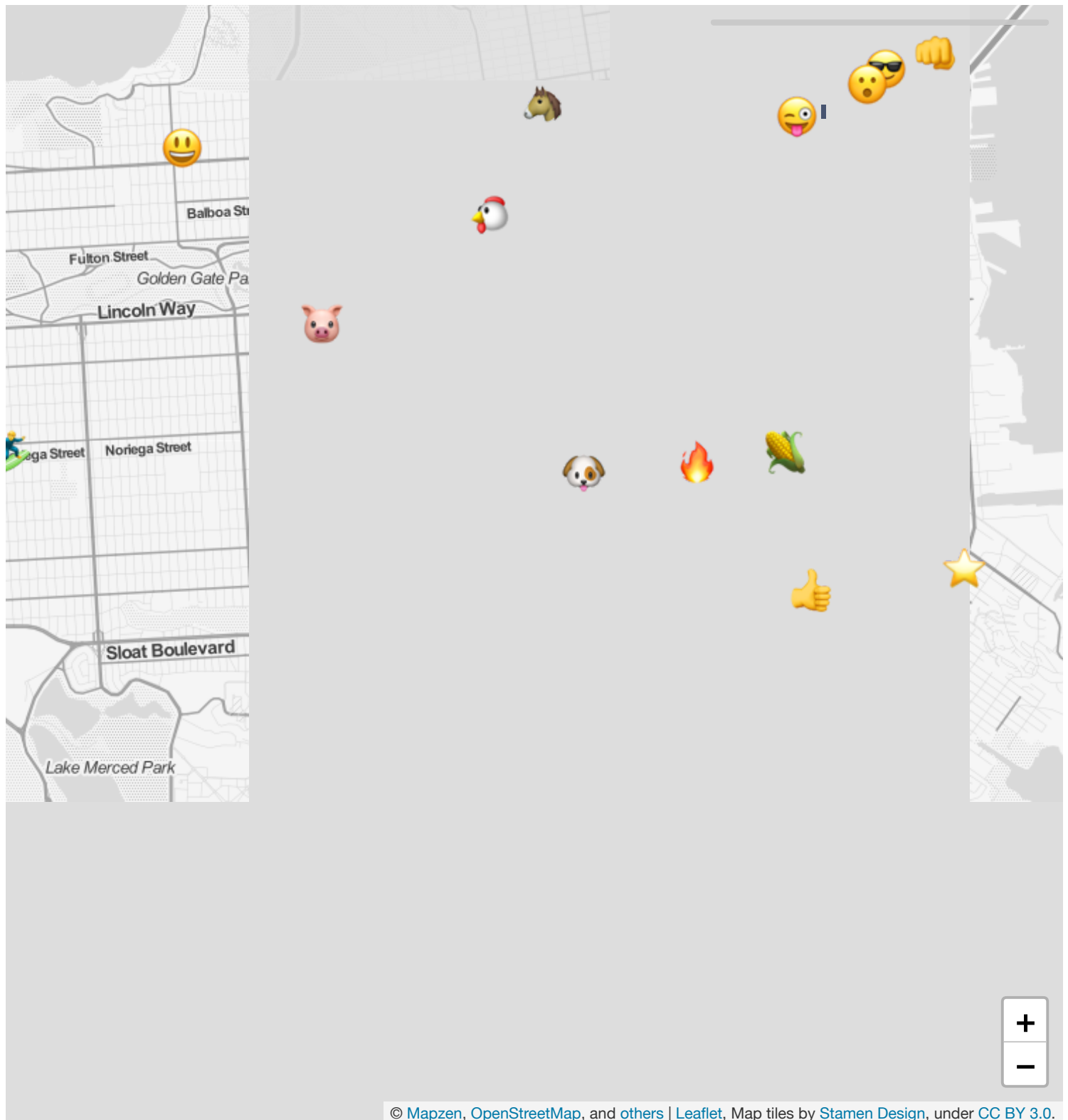
mobility (/tag/mobility)

We're all looking for the most efficient route to get from point A to point B. But what if you add points C, D, E, F, G, and H? Many know this as the "**Travelling Salesman Problem** (https://simple.wikipedia.org/wiki/Travelling_salesman_problem)" or "TSP". We're happy to announce the Mapzen Optimized Route service: powerful enough for businesses and logistics companies yet still accessible to the individual user. Optimized Route is a part of the **Mapzen Mobility** (<https://mapzen.com/blog/introducing-mapzen-mobility/>) toolkit, a set of services powered exclusively by open-source software and open data.

Create your own optimized route

Optimized routes have long been a mainstay in the world of logistics and shipping. Businesses need it too: think of pizza delivery, sales people, and real estate agents showing houses. But wouldn't it be nice if you, the everyday human running errands, had an easy-to-use service that calculates your route based the times and distances to all of the places **you** need to go every day? With Mapzen Optimized Route, anyone with things to do and a list of locations can take advantage of it.

In San Francisco, there are a *lot* of burritos. Too often you ask yourself how you might visit all of the best taquerias in the City. How could one possibly plan this journey of burrito enlightenment? Mapzen Optimized Route is here to help! We here in the Burrito Research Division of Mapzen have geocoded some of San Francisco's top rated burritos according the famed (but sadly retired) **Burritoeater** (<http://www.burritoeater.com/taquerias.php?order=omr>).



Click to go bur-routing full screen (</resources/projects/turn-by-turn/optimized-routing/>) or choose the **walking tour (</resources/projects/turn-by-turn/optimized-routing/walk.html>)**

You may not agree with all of these burrito dispensaries, so drag and drop the emoji across town and watch the route optimize on the fly!

Under the hood – how does it work?

We take your location list and submit it to our cost matrix source-to-target algorithm to calculate the times and distances from each location to every other location. This is the same cost matrix algorithm that we use for our **Mapzen Matrix** (<https://mapzen.com/blog/matrix/>) service and has proven to be extremely efficient.

Under the hood, the cost matrix performs a backward search from all target locations and a forward search from all source locations. The connections between the two search spaces are checked during the forward search. (Any locations that are the same get set to a zero time, distance and are not added to the remaining location set.) We then send the list of times and distances to our optimizer which returns a route response based on the reordered locations.

How do I use this amazing time-saving service?

You can either end your route at the final destination location or you can end your route at the origin where you started. (Please remember that the first and last location will always remain fixed in your route.)

To format an Optimized Route request, we have provided a new action called `optimized_route`. Just send a list of lat/lon locations in any order, your costing mode (auto, bicycle or walk), API key, and Mapzen will take care of the rest! (You can either end your route at the final destination location or you can end your route at the origin where you started. And please remember that the first and last location will always remain fixed in your route.) We will reorder your locations in an optimized route response with a route shape, saving you time and money.

A sample Optimized Route request:

```
https://matrix.mapzen.com/optimized_route?json={"locations":[{"lat":40.78821,"lon":-73.97}
```

You can read the **Optimized Route documentation** (<https://mapzen.com/documentation/optimized>) for more information about the options and **sign up for an API key** (<https://mapzen.com/developers/>). If you want to test your own points, take a look at our **Optimized Route Test Utility** (http://valhalla.github.io/demos/optimized_route/index.html)

A small note on performance and limitations

Performance is fairly similar to our `many_to_many` matrix requests. As mentioned above, we are now using a more efficient cost matrix algorithm that has improved performance substantially. We currently have a few location and distance limits in place for this purpose and you can read more about them in the **documentation** (<https://mapzen.com/documentation/overview/rate-limits/#mapzen-optimized-route>).

Check it out!

Reach out (<mailto:routing@mapzen.com?subject=Optimized%20Route>) if you have questions or suggestions! Sign up for an API key on the **Mapzen Developers page** (<https://mapzen.com/developers/>) and **take a look at the documentation** (<https://mapzen.com/documentation/optimized>).

Note: This post was updated on May 22, 2017 to reflect new documentation URLs and rate limit information.

· 04 August 2016 ·



Kristen DiLuca

Kristen is a software engineer specializing in our routing API services. She also enjoys dabbling in javascript for our open source routing test tools and always welcomes new challenges.



John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.



Hanbyul Jo

Hanbyul does front-end that makes maps and candies.

Access denied...well, not in my country

routing (/tag/routing)

Access denied? Not necessarily. Pedestrians are allowed to utilize **cycleways** (<http://wiki.openstreetmap.org/wiki/Tag:highway%3Dcycleway>) in certain countries.



Photo by Heidi Knisely

Valhalla already uses administrative information to determine if we drive on the right or left side of the street. We also use it to determine when we cross country borders and to set penalties and costs when forming international route paths. Additionally, we recently added support for

country specific access restrictions

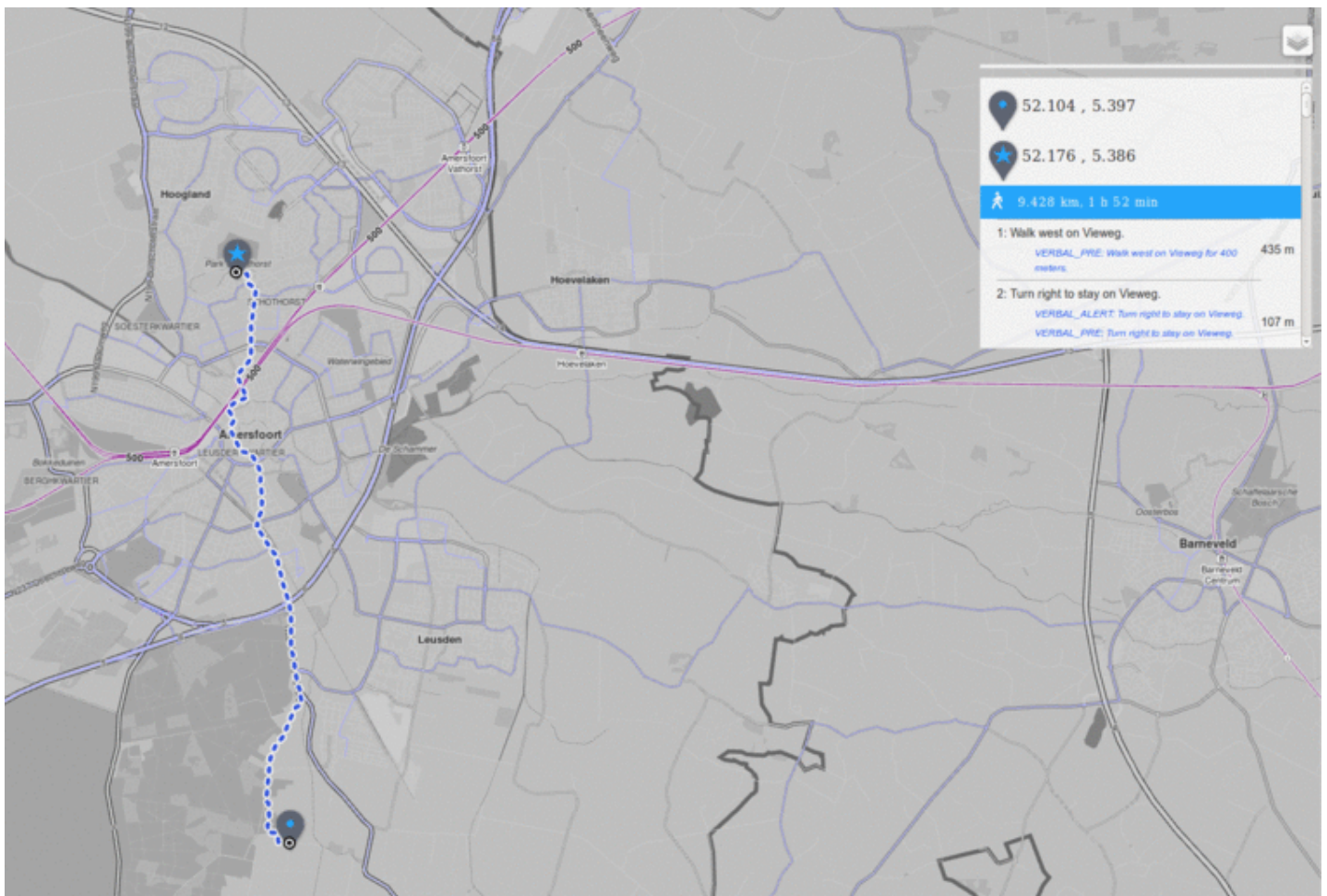
(https://wiki.openstreetmap.org/wiki/OSM_tags_for_routing/Access-Restrictions).

Some countries allow you to walk on **trunk roads**

(<http://wiki.openstreetmap.org/wiki/Tag:highway%3Dtrunk>) but others don't. While the default access restrictions apply to most countries, adding country-specific access restrictions can only enhance the routing graph and deliver more useful routing results.

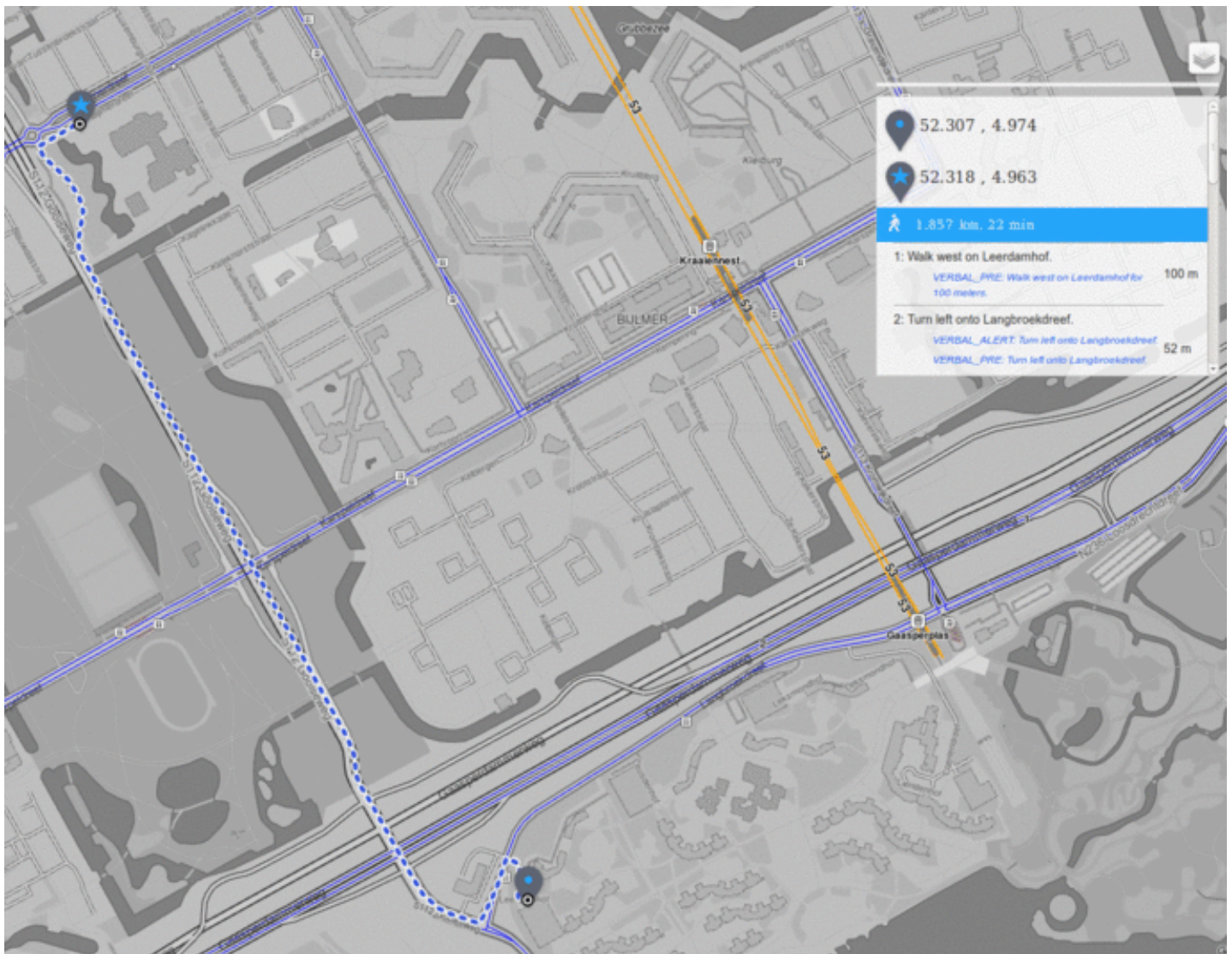
Pedestrians permitted on cycleways

I know a lot of cyclists cringe at the thought of pedestrians on cycleways. However, in the Netherlands it is fine to walk on them — in fact, a Dutch user created an issue showing that we were avoiding cycleways. In the GIF below you can see the original pedestrian route avoiding cycleways, but in reality no one would have used this ridiculous route. Now that we are using **the data from Netherlands access table** (https://wiki.openstreetmap.org/wiki/OSM_tags_for_routing/Access-Restrictions#The_Netherlands) to override the default access logic, you can see that a cleaner and more realistic pedestrian path is calculated utilizing cycleways.



Pedestrians ~~permitted~~ *not permitted* on trunk roads

By default, pedestrians are permitted on trunk roads but in some countries this isn't the case, and is often very dangerous. In the example shown below, the trunk road **Gooiseweg** (<http://www.openstreetmap.org/way/335074126>) has a speed limit of 70 km/hr which is not 100% safe for pedestrians. The updated and safer route takes residential roads and the **Kelbergenpad cycleway** (<http://www.openstreetmap.org/way/115080373>), a safe and legal pedestrian route.



How did we make it happen

Mjolnir (<https://github.com/valhalla/mjolnir>) pulverizes data into a usable form. Well, not really, but it does give us the option to apply attribution to the Valhalla routing data. An administrative database is created via `valhalla_build_admins` and can then be used to determine country for each edge during the building of the graph data. Using the tables on the **country specific access restrictions**

(https://wiki.openstreetmap.org/wiki/OSM_tags_for_routing/Access-Restrictions) page we generate a simple country-to-access array when access differs from the defaults. This data is then stored in the administrative database and accessed during the graph enhancement phase when building the route tiles. If an edge is located within a country with enhanced access, we will check to see if a user of the OpenStreetMap community set the access on this edge. If so, we will not override access with the country updates, otherwise, country-specific access wins.

What's next for administrative information in Valhalla?

While updating the **country specific access restrictions**

(https://wiki.openstreetmap.org/wiki/OSM_tags_for_routing/Access-Restrictions), we realized that we need to ingest the **motorroad key**

(<http://wiki.openstreetmap.org/wiki/Key:motorroad>) and extend the access attribution a bit more. Moreover, we need to enhance the routing graph with country level speeds.

Of course, we would love to process access for more countries; however, there are only a handful of countries listed on the **page**

(https://wiki.openstreetmap.org/wiki/OSM_tags_for_routing/Access-Restrictions). If your country is not listed, please add it and let us know and will get it added into Valhalla ASAP!

As always...check out the **Mapzen Turn-by-Turn documentation**

(<https://mapzen.com/documentation/turn-by-turn/>) and **let us know if you have any questions** (<https://twitter.com/intent/tweet?text=@Mapzen%20@ValhallaRouting%20Hi!>).

· 09 August 2016 ·

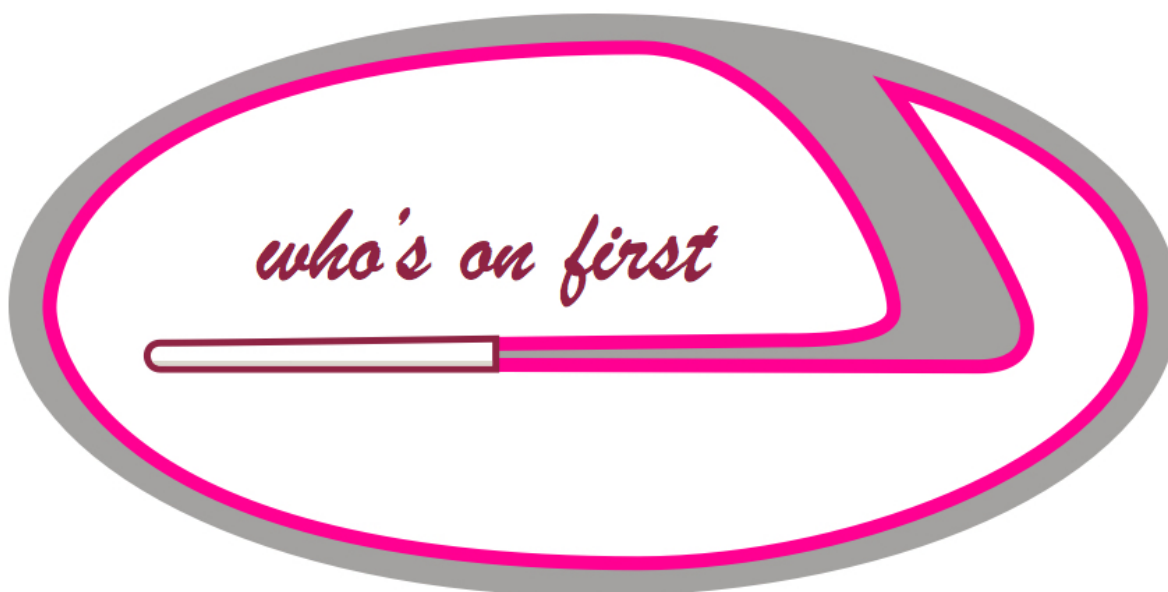


Greg Knisely

Greg is a data munger and software engineer for Mapzen's routing software.

Mapping with Bias

whosonfirst (/tag/whosonfirst)



Mapping With Perspective / April 2016

Aaron Straup Cope / Editor at Large, Mapzen

The following are the slides and notes for a talk I presented at the **Mapping With Perspective** (<https://mapzen.com/blog/mapzen-sf-event-april-26/>) event held at the **Mapzen West** (<https://whosonfirst.mapzen.com/spelunker/id/907212541/>) offices, in April 2016.

If you've read the first **introductory blog post** (<https://mapzen.com/blog/who-s-on-first/>) about **Who's On First** or **the talk I did at FOSS4G** (<https://mapzen.com/blog/spelunker-jumping-into-who-s-on-first>) introducing the Spelunker much of what follows will be familiar. It was a short talk so rather than get lost in the **technical details** (<https://www.github.com/whosonfirst/whosonfirst->

docs/) *I tried instead to focus on some of the principles, and statements of bias, that influence the project. Why we're doing this, rather than how, particularly for people who might not have read those first two blog posts.*

My talk was titled "Mapping with Bias" and this is what I said.

I had stickers made for the **Who's On First** (<https://whosonfirst.mapzen.com/>) project recently. They look **like this**. (<https://mapzen-assets.s3.amazonaws.com/images/mapping-with-bias/stickers.jpg>)

No one has any idea what they depict and that's led to some hilarious speculation about what they "are" ranging from a hockey stick (I am from Canada) to the Lexus car logo to **an e-cigarette** (<https://twitter.com/alloftheplaces/status/720359919749169152>).

It's actually a **pitot tube** (<https://www.grc.nasa.gov/WWW/k-12/VirtualAero/BottleRocket/airplane/pitot.html>). Wikipedia **describes a pitot tube** (https://en.wikipedia.org/wiki/Pitot_tube) as:

...a pressure measurement instrument used to measure fluid flow velocity.

It goes on to describe "flow velocity" as:

...a vector field which is used to mathematically describe the motion of a continuum.

Who's On First (<https://whosonfirst.mapzen.com>) is *not really* a pitot tube but I like that idea that there might be an instrument to measure the motion – the velocity – of people's understanding of place.

Also, I like shiny things (<https://mapzen-assets.s3.amazonaws.com/images/mapping-with-bias/stickers.jpg>).



whosonfirst.mapzen.com **#theory**

Who's On First (<https://whosonfirst.mapzen.com>) is a gazetteer. A gazetteer is a “big bag of places” in which every place has a stable and permanent identifier, supporting metadata and pointers to other IDs in the gazetteer for places with which it has a relationship.

Over **15, 000 words** have written about **Who's On First (<https://whosonfirst.mapzen.com/#theory>)** so far because it turns out that gazetteers are a pretty complicated subject.

Rather than trying to squeeze **all the details (<https://mapzen.com/tag/whosonfirst/>)** in to a 20-minute talk I am going to focus instead on some of the first principles motivating the project and governing our day-to-day work.



(<https://collection.cooperhewitt.org/objects/18297119/>)

The toxic trinity of “geo” has always been the unholy union of: *licensing and coverage and quality*.

The aim of **Who’s On First** (<https://whosonfirst.mapzen.com>) is to tackle all three at the same time.

If and when we are forced to “pick two”

(https://en.wikipedia.org/wiki/Project_management_triangle#.22Pick_any_two.22) we will choose licensing and coverage, so that in the end there is always something left over that people can improve as time and circumstances permit.

These decisions make **Who’s On First** (<https://whosonfirst.mapzen.com>) both ambitious and daunting so I have always felt that it is important for us to have a governing bias with which to negotiate the complexities and the quicksand that the project will inevitably yield.

In many ways, these are principles to help us understand what the project is *not* and to help us understand how things *should be* even if the technology doesn't always work as well as we imagine it should, yet.

A gazetteer is a pretty brainy project. Gazetteers are one of those things that don't seem important at all until they are, at which point they suddenly take on an outsized importance. This means we need to design things in such a way that our work can outlast people's reluctance.

We need to build something with the patience and the stamina, conceptually and financially, to sit quietly in the corner and be ready to be of service when you are and not before.

What follows are six “umbrella” ideas that we keep in mind as we work towards that goal.



a gazetteer

of consensual hallucinations

The first is the idea that **Who's On First** (<https://whosonfirst.mapzen.com/>) is a gazetteer of consensual hallucinations.

“A gazetteer of consensual hallucinations”



(<https://collection.cooperhewitt.org/objects/68775917/>)

The history of geospatial technologies has for most part been one of **force projection** (<http://www.aaronland.info/weblog/2012/03/13/godhelpus/#sxaesthetic>) and **tax collection** (<http://www.aaronland.info/weblog/2015/02/24/effort/#holodeck>). Some people argue they are the same thing.

But there is a good reason that, for example, California alone **has seven state planes** (http://www.conservation.ca.gov/cgs/information/geologic_mapping/state_plane): There are actual tax dollars, and services like emergency responders, that depend on being able to precisely and accurately locate a thing in a world where latitude **can not be neatly subdivided in to equal units** (https://en.wikipedia.org/wiki/Geographic_coordinate_system#Complexity_of_the_problem) across the surface of the globe.

It is worth remembering that coordinate space is one of the truly great abstractions. Being able to reduce the problem, in so many cases, to fit a Cartesian grid has made some pretty amazing things possible.

On the other hand, I cut my teeth working on geo at Flickr where we learned over and over and over again that **no one thinks of place as coordinate data** (<http://code.flickr.net/category/geo>) at least not when it comes to **their photos** (<https://www.flickr.com/places/>).

It's not just Flickr. There is a long and growing list of companies – really, *all* companies – whose services are implicitly built around social rather than administrative notions of place. And social notions of place are messy and an inexact and complicated. This is the space where **Who's On First** (<https://whosonfirst.mapzen.com>) sits.

Who's On First (<https://whosonfirst.mapzen.com>) is, by design, not a gazetteer of unitary perspective.



all the places

or: “multiple geometries”

Or put another way, **Who's On First** (<https://whosonfirst.mapzen.com/>) is **not a gazetteer of geometries**.



“Multiple geometries”

(<https://collection.cooperhewitt.org/objects/18643589/>)

One of our earliest decisions was on that each record in **Who's On First** (<https://whosonfirst.mapzen.com>) would contain multiple geometries.

As a rule every place in **Who's On First** (<https://whosonfirst.mapzen.com>) should contain a so-called “ground truth” geometry. A ground truth geometry is like Benoit Mandelbrot's **map of England**

(https://en.wikipedia.org/wiki/How_Long_Is_the_Coast_of_Britain%3F_Statistical_Self-Similarity_and_Fractional_Dimension) which, by definition, means it will always grow in size and detail.

Some places might even have *two* ground truth geometries, one clipped to the coastline and another than includes territorial waters. These geometries are especially good **for reverse geocoding** (<https://github.com/whosonfirst-data/whosonfirst-data/issues/367>) but the salient point in all of this is that the geometries themselves, as often as not, enforce the biases of their use.

There might also be “folk” geometries which encode **a common (or folk) understanding of a place** (<https://github.com/whosonfirst-data/whosonfirst-data/blob/master/data/859/225/83/85922583-alt-mapzen.geojson>) rather than an official designation. Think of Eric Fischer’s **Locals and Tourists maps** (<http://www.moma.org/interactives/exhibitions/2011/talktome/objects/146200/>).

There will inevitably be disputed geometries. These are different from places that have been classified as **disputed** (<https://whosonfirst.mapzen.com/spelunker/placetypes/disputed/>), places like Kashmir or the Golan Heights. These are places where the stakes are not so high. Places where even though their may be officially recognized boundaries people still bicker over the details. Effectively **all neighbourhoods** (<http://www.gowanusheights.info>), everywhere.

We may disagree on where **The Tenderloin** (<https://whosonfirst.mapzen.com/spelunker/id/85865903/descendants/?exclude=nullisland&placetype=microhood>) starts and stops, for example, but we all agree that The Tenderloin *exists*.

The purpose and the value of **Who’s On First** (<https://whosonfirst.mapzen.com>) is in giving those **notions of place** (<https://whosonfirst.mapzen.com/spelunker/id/102112179/>) a collective proof. In giving them a mass and weight and a gravity in the universe that other people and products can orbit.

common ancestors

because all families are psychotic

Another core principle of **Who's On First** (<https://whosonfirst.mapzen.com>) that is every record shares **a common set of ancestors**.



(<https://collection.cooperhewitt.org/objects/18643587/>)

Hierarchies, in particular administrative hierarchies, vary wildly from country to country. We used to say that all locations in **Who's On First** (<https://whosonfirst.mapzen.com>) share a common hierarchy but I think that was often more confusing than not.

It is an articulation that lends itself to the idea, incorrectly, that there is a single comprehensive hierarchy which encodes all the relationships between places. That is not what **Who's On First** (<https://whosonfirst.mapzen.com>) tries to do.

Instead we have said that there are **five common placetypes** (<https://github.com/whosonfirst/whosonfirst-placetypes#here-is-a-pretty-picture>) – *continent, country, region, locality and neighbourhood* – and that every record in **Who's On First** (<https://whosonfirst.mapzen.com>), regardless of its **specific placetype** (<https://whosonfirst.mapzen.com/spelunker/placetypes/>) has at least one of the common placetypes as an *ancestor*.

This acts as a baseline for a global dataset, both on a conceptual and a practical level. It is important to us that, within reason, we not impose any single architectural approach or set of technical requirements in order to be able to use **Who's On First** (<https://whosonfirst.mapzen.com>).

Five “database” columns for encoding a global hierarchy seems like a reasonable trade-off in 2016. If you need to include **Brooklyn, NY** (<https://whosonfirst.mapzen.com/spelunker/id/421205765/>) (which is technically a **borough** (<https://whosonfirst.mapzen.com/spelunker/placetypes/borough>)) in your dataset then you'll need to add a sixth column but that's your business. Otherwise you can hopefully make do with **New York City** (<https://whosonfirst.mapzen.com/spelunker/id/85977539/>).

Importantly, unknown place types are not a fatal error. They are left to the needs and discretions of people using **Who's On First** (<https://whosonfirst.mapzen.com>) for whatever they need to use it for, without sacrificing a common ground where all of these projects can still comfortably hold hands.

There is also a related discussion about places having *multiple* hierarchies but we don't have time for that tonight. Suffice it to say that places **can and do have multiple hierarchies** (<https://github.com/whosonfirst/whosonfirst-placetypes#hierarchies>) for much the same reasons that a place might have multiple geometries.



all the ancestors

all of this has happened before

Who's On First (<https://whosonfirst.mapzen.com>) is not a linear scorched-earth view of the world.



"All of this has happened before"

(<https://collection.cooperhewitt.org/objects/18334353/>)

Places change. The physical boundaries of the USA **changed 141 times**

(<https://github.com/whosonfirst-data/whosonfirst-data/issues/176>) between the years 1789 and 1959. The entire notion of what Yugoslavia *meant* changed three times in the 20th century before finally atomizing in to **seven countries**

(https://en.wikipedia.org/wiki/Yugoslavia#/media/File:Former_Yugoslavia_2008.PNG), by 2008.

Ultimately there is a much larger question about how an individual, or worse a community, decides whether an event constitutes a simple update versus a fundamental change. This is the realm of hard philosophical questions and those are things we are not going to try to answer.

We can provide breadcrumbs, though. Every record in **Who's On First**

(<https://whosonfirst.mapzen.com>) has both a `superseded_by` and `supersedes` property that are used to signal that a change has occurred but *not necessarily why*. That part is left up to you.

These properties act as a kind of **linked-list for places**

(https://en.wikipedia.org/wiki/Linked_list) indicating, for example, that the Kingdom of Yugoslavia was superseded by the Federal People's Republic of Yugoslavia in 1946, and so on.

This decision means two things:

1. That there might be **multiple entries**

(<https://whosonfirst.mapzen.com/spelunker/search/?name=stamen%20design>) for the “same” place in **Who's On First** (<https://whosonfirst.mapzen.com>) and consumers of the data need to account for this fact.

2. That if you have been using the **the first iteration**

(<https://whosonfirst.mapzen.com/spelunker/id/571704337/>) of a place in **Who's On First** (<https://whosonfirst.mapzen.com>) its *meaning and semantics* won't suddenly change when there is a **legitimate reason**

(<https://whosonfirst.mapzen.com/spelunker/search/?name=stamen%20design>) to create a **second iteration**

(<https://whosonfirst.mapzen.com/spelunker/id/907212647/>).

We do this as a way to foster confidence in the robustness and durability of **Who's On First** (<https://whosonfirst.mapzen.com>) identifiers. The past is complicated territory and though it is not the focus of our daily work we want to try and make sure that it is always welcome.

reflect debate

a gazetteer of signal fires

Who's On First (<https://whosonfirst.mapzen.com>) is a gazetteer of signal fires.



(<https://collection.cooperhewitt.org/objects/18334351/>)

It's probably obviously by now but it bears repeating: The world is full of complex and contradictory opinions. We do not want to try and settle those debates. We can not settle those debates.

For almost as long as we've had the notion of place itself people have had the benefit of complete sentences and entire paragraphs and even book-length arguments to make sense of the nature and meaning and value of place.

And still we don't agree so I don't know why anyone can imagine that a bag of key/value pairs will do better at answering any of these questions.

Obviously there are a few instances where **Who's On First** (<https://whosonfirst.mapzen.com>) needs to assert some degree of editorial opinion about but as a rule we try to do this as infrequently and as transparently as possible.

When there is genuine debate about something we leave it to the consumers of the data to interpret. We want to signal that *there is debate* about something rather than try to gloss over the awkward bits.

failure scenarios

the data is not the database

Finally, **the data is not the database.**



(<https://collection.cooperhewitt.org/objects/18616235/>)

I mentioned at the beginning that **Who's On First** (<https://whosonfirst.mapzen.com>) was designed to “outlast people’s reluctance”.

What this means is that **Who's On First** (<https://whosonfirst.mapzen.com>) is not optimized for any one application including **Mapzen** (<https://www.mapzen.com/>), which makes for some awkward conversations around the office from time to time.

What this means, in concrete terms, is that at its core **Who's On First** (<https://whosonfirst.mapzen.com>) is a **gigantic bag of plain-text files** (<https://whosonfirst.mapzen.com/data/>). The failure scenario for updating a **Who's On First** (<https://whosonfirst.mapzen.com>) record should always be the ability to edit it using nothing more than a text editor. You shouldn't *have* to do that but when everything else breaks you still *can* do that.

The point is not that **Who's On First** (<https://whosonfirst.mapzen.com>) doesn't play with databases but that it should be able to play nicely with *all* the databases. The point is that the demands **Who's On First** (<https://whosonfirst.mapzen.com>) places on its users should be as universal as possible across platforms and concerns.

Sometimes this makes getting things set up a little harder than we'd like but it's 2016 and we've all gotten pretty good at **processing text files at scale** (<https://dl.acm.org/citation.cfm?id=512948>) and feeding them in to databases.

Despite all the advances we've made over the years it turns out that the simplest, most universal and accessible thing is still plain-old, plain-vanilla, plain-text files on disk.

They have the added benefit of being (still) the most reliable way to archive things **as the technological landscape shifts** (<http://www.cooperhewitt.org/2013/08/26/planetary-collecting-and-preserving-code-as-a-living-object/>), year over year. We can **print them out** (<http://booktwo.org/notebook/wikipedia-historiography/>), if necessary.

This focus – of demanding a high degree of portability and durability in our work - is very much influenced by **the early systems designs** (<https://www.princeton.edu/%7Ehos/frs122/unixhist/finalhis.htm>) for the Unix, and Multics before it, operating system and more recently the **Unicode** (<http://unicode.org/>) project.

These are **subjects** (<https://www.bell-labs.com/usr/dmr/www/hist.html>) that could occupy many, many more nights of presentations **all on their own** (<http://www.unicodeconference.org/>) and it remains to be seen whether we can accomplish our work as well as they did theirs.

But that is the work.

thank you

whosonfirst.mapzen.com
@alloftheplaces

Thank you. If you'd like a sticker **send up a flare** (<https://www.twitter.com/alloftheplaces>).

All of the images in this presentation are part of the collection of the **Cooper Hewitt Smithsonian Design Museum** (<https://collection.cooperhewitt.org/>). They are:

- **Bound Print (France); Purchased for the Museum by the Advisory Council; 1921-6-559-19**
(<https://collection.cooperhewitt.org/objects/18297119>)
- **Paper Construction, Buck Rogers, 25th Century Featuring Buddy and Allura in "Strange adventures in the Spider Ship"; Attacked by the Giant Reptile, ca. 1935; Collection of Smithsonian Institution Libraries**
(<https://collection.cooperhewitt.org/objects/68775917>)
- **Figure, ca. 1960; porcelain, enamel; Gift of Ludmilla Shapiro; 1993-13-2**
(<https://collection.cooperhewitt.org/objects/18643589/>)
- **cosmonauts and rocket Figure, 1960-70; Made by Gzhel Porcelain Factory ; porcelain, enamel.; Gift of Ludmilla Shapiro; 1993-13-1** (<https://collection.cooperhewitt.org/objects/18643587/>)
Figure; biscuit; Gift of Eleanor and Sarah Hewitt; 1931-88-89-a/d
(<https://collection.cooperhewitt.org/objects/18334353>) **Triton Figure, 19th century; Manufactured by Sèvres Porcelain Manufactory (France); France; biscuit porcelain; Gift of Eleanor and Sarah Hewitt; 1931-88-88-a/d**
(<https://collection.cooperhewitt.org/objects/18334351/>) **Ink Plot; computer ink plot; 1981-19-1**
(<https://collection.cooperhewitt.org/objects/18616235/>)

· 15 August 2016 ·



Aaron Straup Cope

Aaron is happiest in the presence of olive oil.

© 2017 Mapzen

Ladders for Leaders, Summer 2016

interns (/tag/interns) **engineering** (/tag/engineering) **search** (/tag/search)

tangram (/tag/tangram)

Thunder roars, lightning claps, and rain pours down from the cloudy sky. Seeing several Mapzen developers gather by the window, I join to gauge the severity of the thunderstorm for myself. In this unlikely moment, I feel a sense of belonging and camaraderie. Sheltered inside, we gaze in awe of the raindrops veering sideways like a dust storm. Wow, I am here, at a tech startup in NYC. The open office space, whiteboard walls covered with diagrams, and wood floors reaffirm my reflection. Everything is even better than what I imagined when I first got an email for a phone interview through the **NYC Ladders to Leaders summer internship program** (<http://www1.nyc.gov/site/dycd/services/jobs-internships/nyc-ladders-for-leaders.page>).

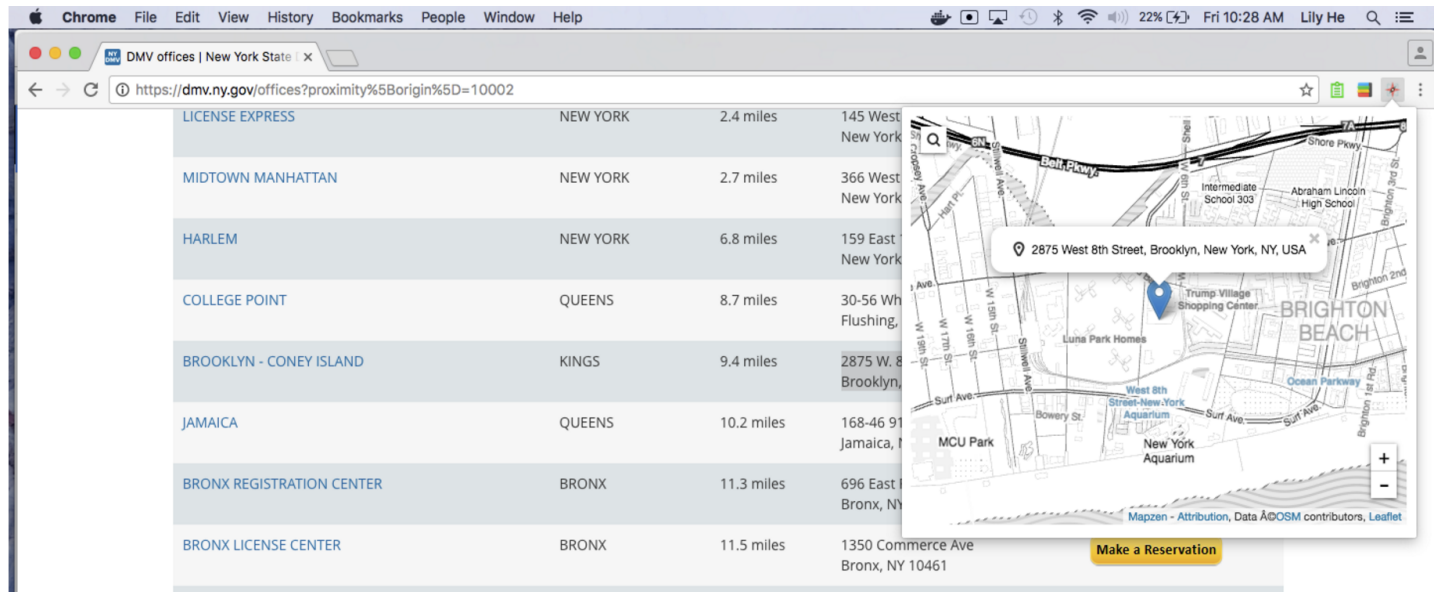
Although I am just an intern straight out of high school, I have been welcomed by everyone at Mapzen. From my day one introduction to lunches to weekly team meetings, I was encouraged to participate, lean in, and absorb as much as I could. Not once did I feel alienated because I am a woman in tech. One of my proudest moments was demoing what I accomplished during the team meeting.

During my time here, I was part of the DevOps crew. My first project was to add a new page to the admin website. The projects page showed charts of API key hits and the different status response codes. While 200 meant OK, there was no way of knowing whether the successful responses were cached or hitting the API servers. Thus, I used **React** (<https://facebook.github.io/react/>), a JavaScript library, to insert a new chart and table specifically for the cached 200 and origin 200 responses. Along the way, I obliterated the admin page custom CSS in favor of using the Mapzen Styleguide.

The second project I focused on was Dockerizing the developer page. **Docker** (<https://www.docker.com/>) is a lightweight and efficient software containerization platform. Instead of creating a separate operating system for app as virtual machines do, Docker containers share an OS using the Docker Engine. With Docker, developers can run and change different parts of the website quickly. My job entailed writing the dockerfile and docker-compose files for various repositories.

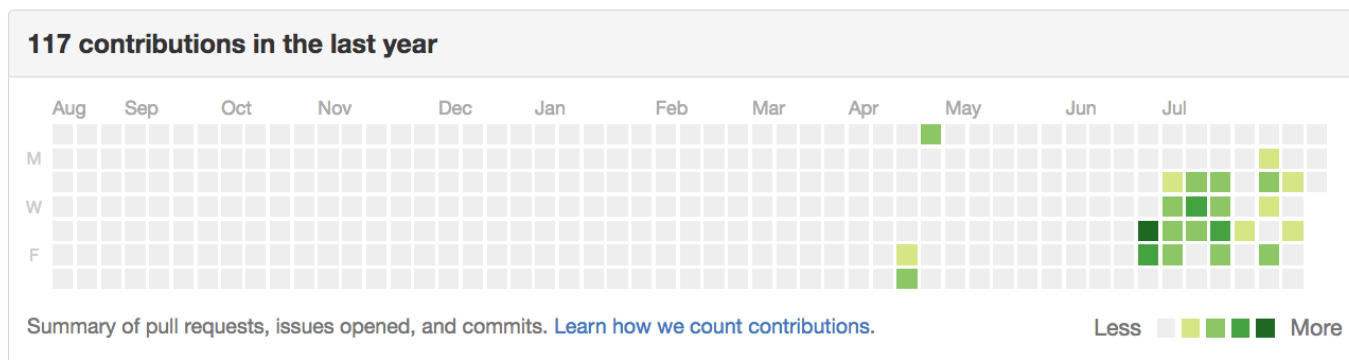
While my projects were small, manageable pieces, I knew I contributed to the community. This experience of working in a company with fifty others was invaluable. A project for a class could not simulate the same effect.

The final phase of my internship was an independent, standalone project. I chose to take advantage of Mapzen.js for its compactness and make a Google Chrome extension.



Nearly all of the map browser extensions on Chrome used Google Maps. Let's change that trend! Using Tangram for map tiles and Pelias for geocoding, **I created a simple add-on** (https://github.com/tigerlily-he/Mapzen_Chrome_Extension) to help users locate a place on a map without opening a new tab. Just highlight the address, city, or landmark and click the extension button to open up a Tangram map built on top of Leaflet. Stay tuned for a more detailed post on my process developing the Mapzen Chrome Extension.

Six weeks ago, my GitHub contribution diagram was completely grey. Aside from the few commits I made during a 24-hour hackathon at QueenHack back in April, I had not touched Git. All I knew how to execute were `git init`, `git add` and `git commit -m`. Now, my contributions count is 166 and a mosaic of green squares decorating the page.



I have learned to rebase, amend and resolve merge conflicts. The terminal does not seem like a dangerous black hole for accidental destruction. Rather, it is a handy and powerful tool to do almost anything.

This summer has gone by incredibly fast. Every day presented different challenges and victories. I actively learned at every moment. For example, the articles shared on the Slack channels enlightened me on **Null Island** (<https://nullis.land/#lat=0.00000&lng=0.00000&z=18>), subway games, and maps from World War II. Of course, I had fun joking around, participating in team events, and being the test user for projects.

Thank you, Evan, for taking me under your wings and guiding me through obstacles. Thank you to Baldur for giving me the chance to smash keys (and code). Thanks to everyone at Mapzen for letting me preview the future of digital map making. Lastly, thank you to **NYC Ladders for Leaders** (<http://www1.nyc.gov/site/dycd/services/jobs-internships/nyc-ladders-for-leaders.page>) for connecting me to Mapzen.

· 17 August 2016 ·



Lily He

Columbia University undergrad, two-time intern at Mapzen (Product and Search teams) via NYC SYEP Ladders to Leaders.

All of the Places

whosonfirst (/tag/whosonfirst) **data** (/tag/data)

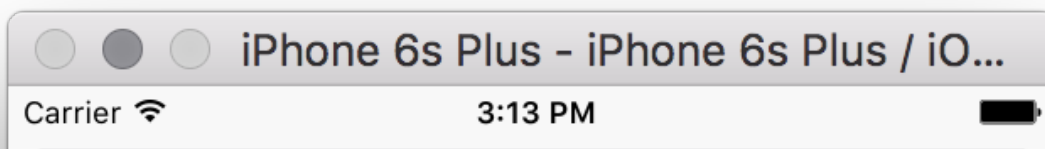
*TL;DR I got lost with my friend, and then found **alloftheplaces.xyz** (<https://alloftheplaces.xyz/>) using all of the Mapzen APIs.*

I was meeting my friend **Caroline** (<https://twitter.com/carolinesinders>) on the July 4th holiday to discuss **a project we're working on** (<https://smalldata.coop/>). She said "we must find something holiday themed," and so we planned a field trip to Ellis Island. I suggested a time and sent her **a link** (<https://www.google.com/maps/place/Ellis+Island+from+Battery+Park>) (now defunct) to where we should meet up. On Monday morning I waited for a while next to the line of tourists, and some time passed. It was a holiday, so there was no particular rush, but I started to suspect she was lost. And then, after some confused texting, we discovered we'd each ended up in different states!

The ferry to the Statue of Liberty and Ellis Island leaves from two different docks. One departs from Battery Park, at the tip of Manhattan. The other leaves from across the Hudson, in New Jersey. I had sent her a URL that yielded ambiguous results. Testing the link afterward suggested that depending on what kind of device you used, where you made your request from, (perhaps various other personalization factors?) it was unpredictable where the map might lead you.

In this case my phone showed **New York** (<https://whosonfirst.mapzen.com/spelunker/id/907288683/>), hers showed **New Jersey** (<https://whosonfirst.mapzen.com/spelunker/id/907289587/>). We sorted out the confusion and met in the middle, on Ellis Island. All was good.

The experience stuck with me. How is it that one URL could yield a single, *but different* result for each person? Further inspection made me realize I'd linked to a photo icon on the map, and that linking to photos isn't well supported on the iOS version of Google Maps.





I'd also complicated the situation by modifying the URL before I shared it. Part of what I removed, the latitude and longitude coordinates, turn out to be an important fallback for settling search result ambiguities. But the link worked for me at the time, so I figured it was safe to share. Nobody is better at intuiting meaning from fuzzy text than Google's search engine. But sometimes it's better for an interactive map to fail rather than offer up a guess. And this appears to be the new behavior, so kudos to the Google Maps team for improving that.

I got to thinking about how our own **Who's On First** (<https://mapzen.com/blog/who-s-on-first/>) gazetteer might help in this type of situation. We already have the **Spelunker** (<https://whosonfirst.mapzen.com/spelunker/id/874411113/>) to find and inspect records, but I decided to build something with far less detail. I just want to *link to a place on the map*, and get directions there. And decorate it with links to other relevant websites—to our own **Spelunker** (<https://whosonfirst.mapzen.com/spelunker/>), to the official web presence for the place (if one exists), to social media, Wikipedia, etc.

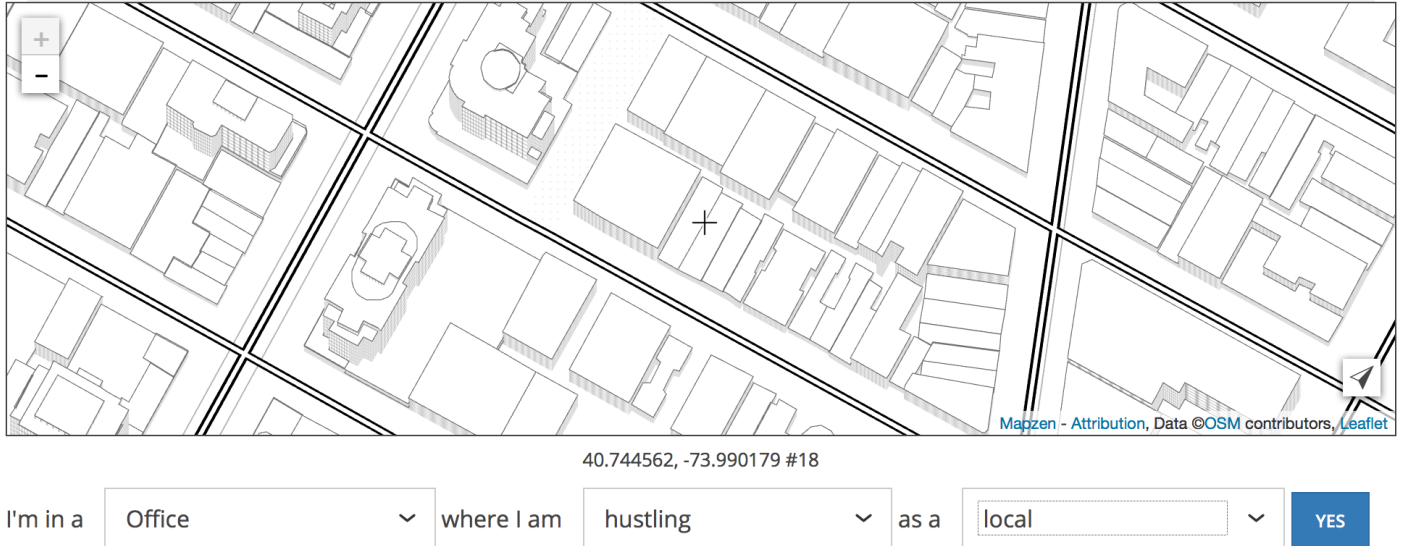
My place-permalinking website is called **alloftheplaces.xyz** (<https://alloftheplaces.xyz/>) and it's still very much an early prototype. As such, it still doesn't quite do what it says on the tin. We are far from including "all of the places," if that is even possible. It's an aspirational name, and is consistent with the Who's On First **Twitter account** (<https://twitter.com/alloftheplaces>).

I wired up **Mapzen.js** (<https://github.com/mapzen/mapzen.js>) using the new **Walkabout** (<https://mapzen.com/blog/walkabout/>) map style, and set it up to pull in data from **Who's On First** (<https://whosonfirst.mapzen.com/>). On top of that: **Turn-By-Turn navigation** (<https://mapzen.com/products/turn-by-turn/>) and **Mapzen Search** (<https://mapzen.com/products/search/>). This little website could just as well be called “All of the Mapzen Services.”

Last week **Aaron** (<http://aaronland.info/>) asked me to take the directions feature out for a spin and to keep track of my experience. So I did that, and it helped clarify some ideas and feelings about using mobile devices to navigate meatspace.

A lot of this may be specific to an NYC-style of routing: transit-focused, with sensitivity to offline use. On the other hand, a lot of general-purpose mapping apps have an everyone-drives-everywhere bias. I'm still undecided about whether a mobile web sloppy map is up to this task, but I do think it's a good way to quickly try out UI ideas.

First issue: Who's On First had no record of the place I was trying to navigate to (Grand Army Plaza). There was also no record of the art space I was traveling to afterward (Pioneer Works, now **added** (<https://alloftheplaces.xyz/907130503>)). The data will get better, and I'd like to experiment with capturing new places directly from alloftheplaces.xyz, perhaps by collecting them via *another* tiny Mapzen service that Aaron is working on called Maplibs.



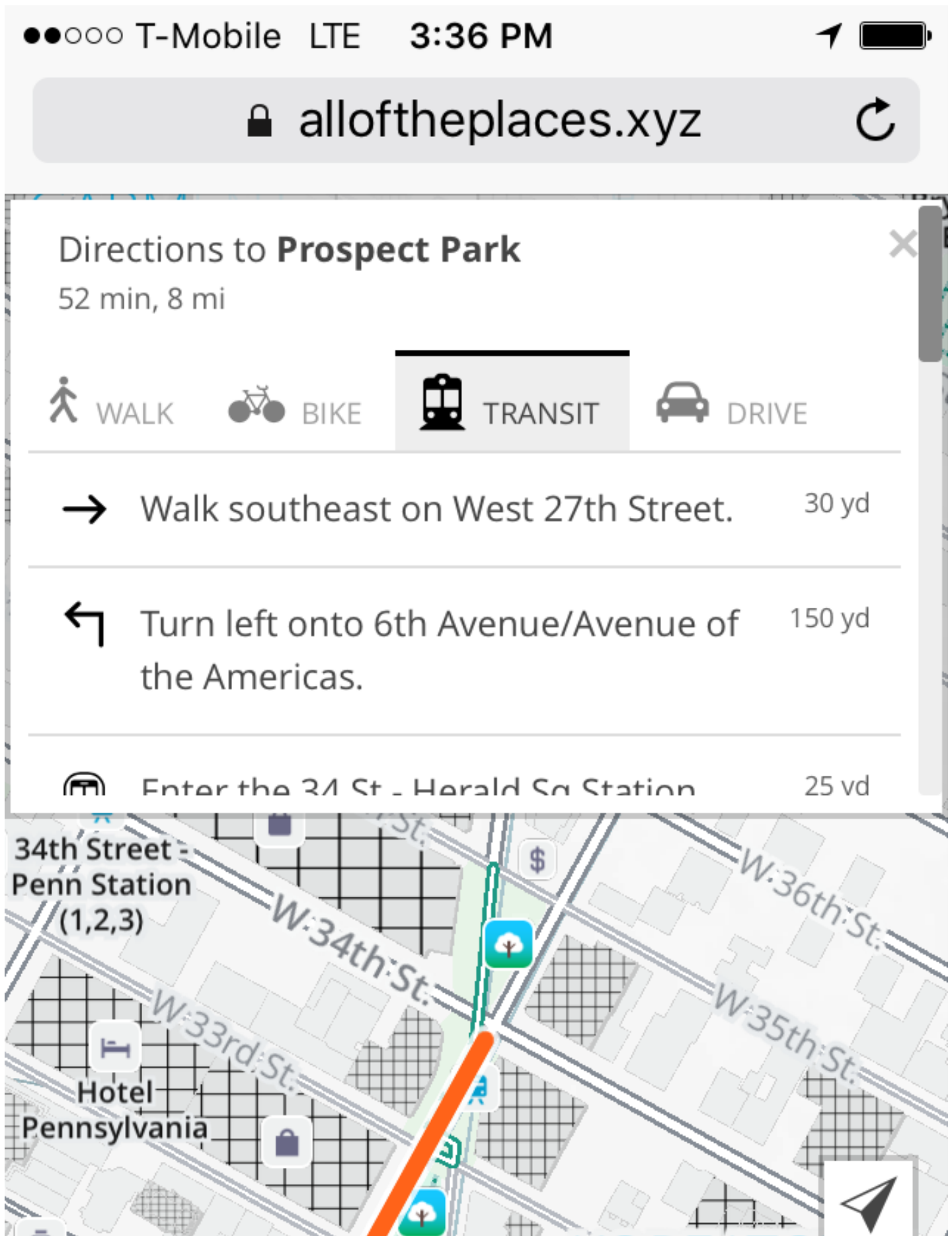
Second issue: As a New Yorker, I have opinions about things like local vs. express trains. When I asked for a route, it only offered me a single choice, though the Mapzen Mobility team has alternate routes on their radar.

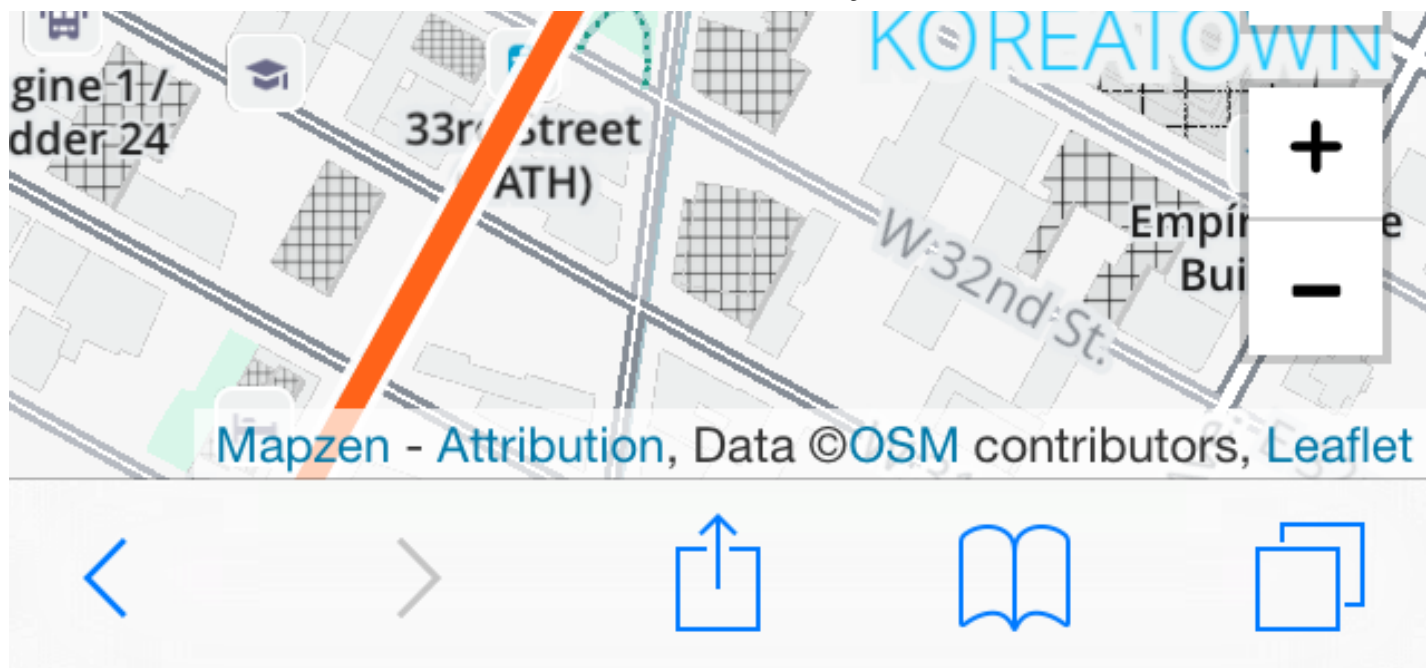
At this point in my test I still haven't left yet, but I'm using the routing service as a probe to figure out how soon I need to leave. The initial view should emphasize the travel time more. A feature that might be useful here is a built-in reminder: "hey, time to start leaving." Apple does this with Calendar.app events with locations, but it seems to assume that everyone drives everywhere.

I leave, and I'm transitioning from my desktop to my phone. I have to look up the trip again on my phone, and I'm probably doing it in the elevator where bandwidth is iffy. A thing that Google Maps does well here is it uses my Google account to sync up recently searched things, which means I don't have to thumb-type my search again. This is a nice feature that feels like an invasive lock-in, but it's still a nice feature.

I'm thinking about how to offer a similar experience but with less of Google's "let us organize *all of your data*." For starters I could save each search query in the web browser cache, so that found things stay found. But the real challenge in a feature like this is syncing these searches between devices and doing it in a way that keeps the user in control of her or his own data. Building out all the plumbing required for that type of use case makes me think of **Carl Sagan's apple pie recipe** (<https://www.youtube.com/watch?v=7s664NsLeFM>).

In any case, here's what I'm seeing on my phone as I leave the office:



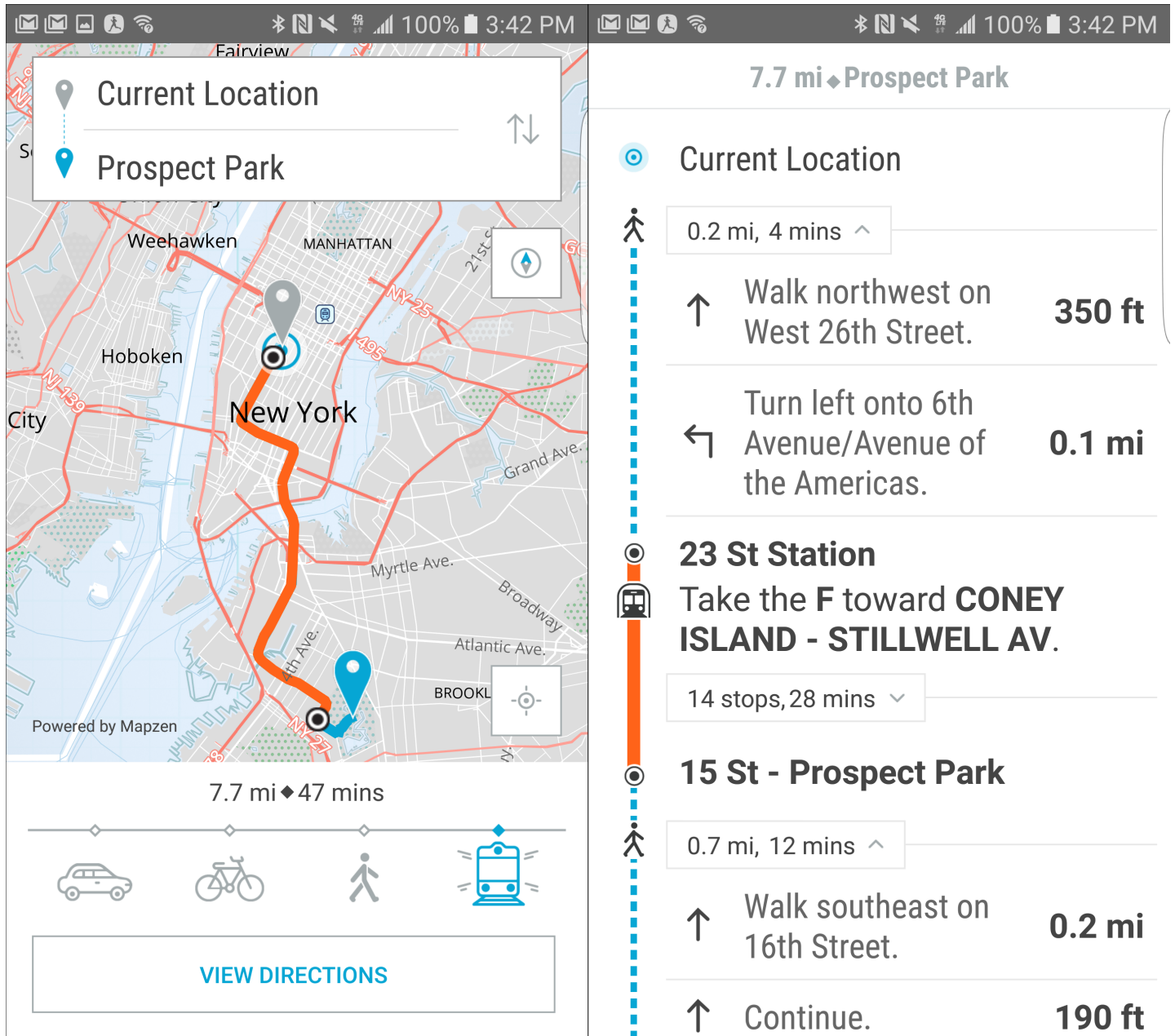


Using this interface on such a tiny screen is still kind of a hot mess. To a large extent this is because I made a compromise to split up the vertical screen space between map and directions. Scrolling through that list of walking directions inside such a tiny box is painful, and offers too much detail.

Maybe what I want is a summarized version that says: + Walk to 34th St - Herald Sq Station + Take the B train to 7th Ave + Walk to Prospect Park

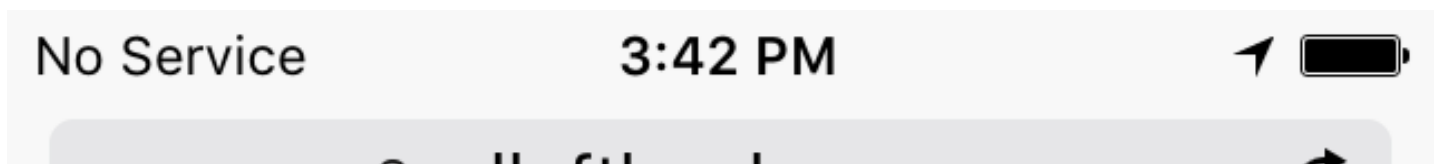
And then I could tap on the + for each of the high-level steps to get more details. Maybe I can use the raw turn-by-turn data and group each of the steps by mode to create each top-level step. Worth a try anyway.

Sarah Lensing (<https://twitter.com/sarahlensing>), who works on the mobile team here, is testing a version of the directions view in Erasemap that takes a similar approach, structuring the various modes of a multimodal trip (available soon!)



Now I'm on the train, and I've lost my bandwidth, but I'm using my phone to figure out which stop to get off at. This is another place where that overview view could be helpful—I really care about “where do I get off the train?” and to a lesser degree “where am I going to walk once I get there?” Bonus points if I could pre-cache a high zoom level neighborhood map of my destination to look at while I'm en route. To the extent that it's possible, anything I download on my phone should be cached for offline use, probably with something like **localForage** (<https://github.com/localForage/localForage>).

Here's how things look now on the train:





Enter the 34 St - Herald Sq Station.

25 yd



Take the B toward BRIGHTON BEACH.
(7 stops)

7 mi



Exit the Prospect Park Station.

150 yd



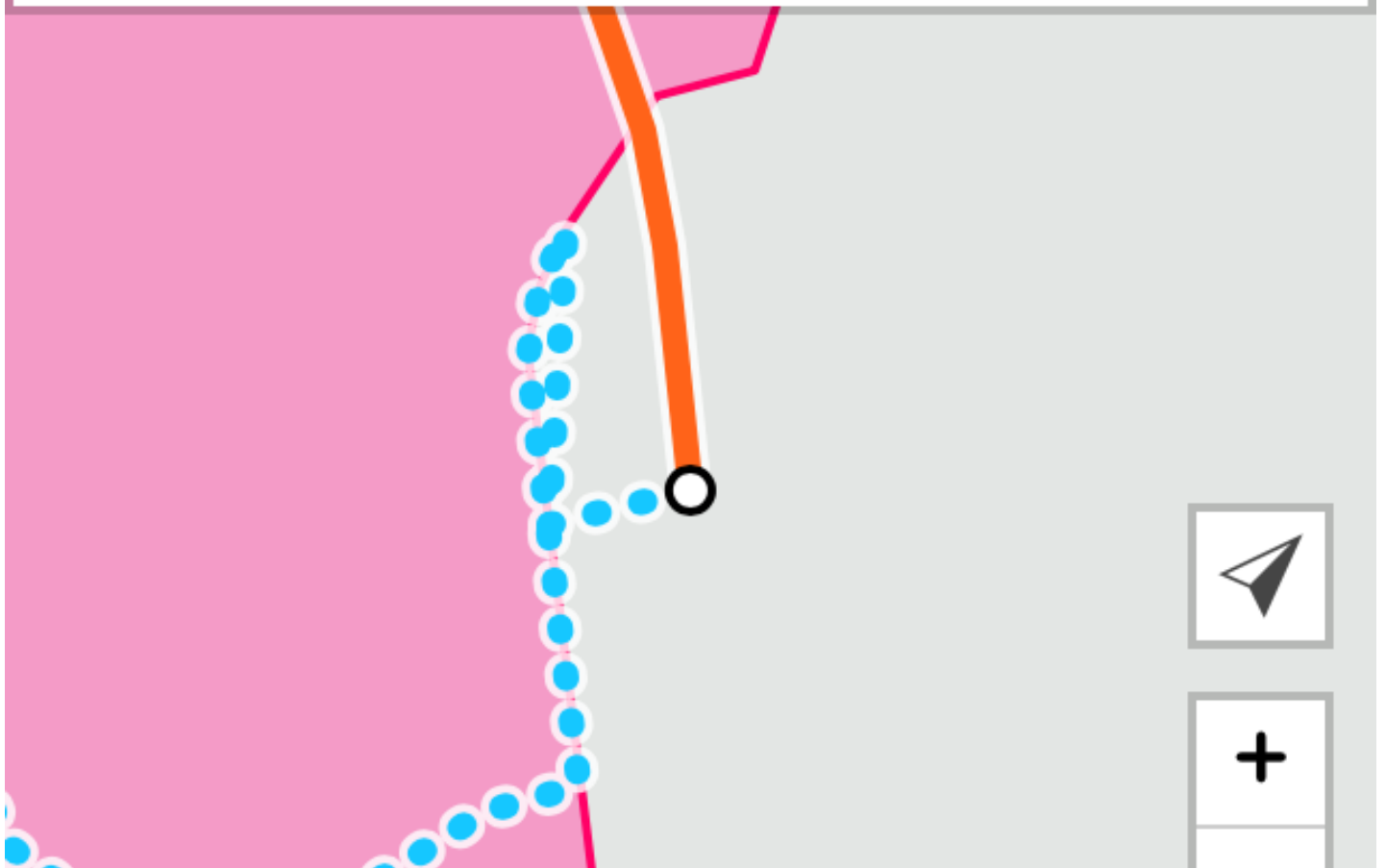
Walk west on East Lake Drive.

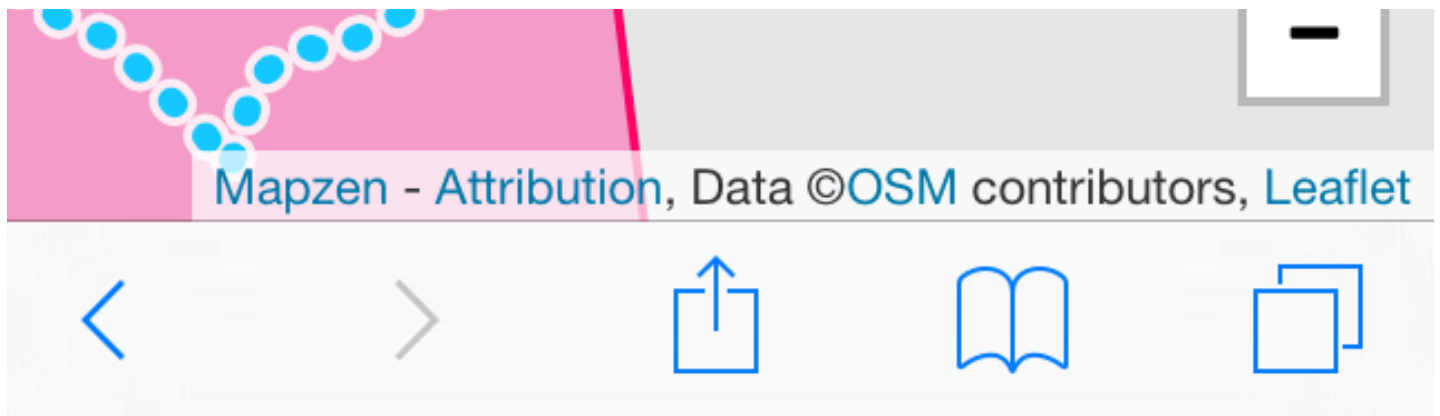
35 yd



Bear right.

10 yd





I make it to my destination and I'm done, but my map is still trying to geolocate me. Was it trying to do this the whole time I was underground? I need some way to tell the map "I'm done" and for the map to notice "we're offline, stop trying to geolocate." I wonder if most browsers/phones are smart enough to detect when you've lost connectivity and stop trying to geolocate you?

Uncertain, but I can always just listen for an **offline event**

(https://developer.mozilla.org/en-US/docs/Online_and_offline_events) so the map responds correctly when I go underground.

Another pain point I've experienced nearing the end of the trip is when I emerge from the train, blinking in the bright sunlight, and the map hasn't really accounted for my new situation. Which way should I walk? If I have a CitiBike membership (I do), where are the racks with available bikes? **Citymapper** (<https://citymapper.com/>) has a great bike sharing interface for this type of scenario), and the Mapzen Mobility team is thinking about **how to implement this** (<https://github.com/transitland/transitland-datastore/issues/310>).



I made it to Grand Army Plaza. The test was a partial success, with lots of pointers toward things that should change. I don't think my experimental web map is going to become anyone's new default, but there's a whole lot you can spin up quickly with various Mapzen APIs. The capacity to add new records to the Who's On First dataset, and then *link to them*, means I'll be far less likely to accidentally send my friends to New Jersey.

· 24 August 2016 ·



Dan Phiffer

Dan is a data carpenter, which tbh is kind of a made up title. But he likes it all the same.

Everything you wanted to know about Mapzen Support but (shouldn't have been) afraid to ask

[tangram \(/tag/tangram\)](/tag/tangram) [search \(/tag/search\)](/tag/search) [vector-tiles \(/tag/vector-tiles\)](/tag/vector-tiles)

[transitland \(/tag/transitland\)](/tag/transitland) [documentation \(/tag/documentation\)](/tag/documentation) [support \(/tag/support\)](/tag/support)

We all know it and we've all been there. The anxiety of **not** knowing something. That dread, that fear that everyone else knows the answer that you can't quite figure out. We hope that's just the stuff of nightmares induced by watching too much Jeopardy!, but sometimes it can be hard to ask for help...or know where to start.

Mapzen wants to make sure you never have to experience this. We offer many ways to learn more about our products and contact us—from self-service documentation to in-person technical assistance. We provide all these options at no cost to you.

Documentation and blog posts

Your first resource for information about how to use Mapzen's products is the **technical documentation** (<https://mapzen.com/documentation/>). Click the Documentation link at the top of any page on Mapzen's site to get to our help system. Documentation includes a variety of content, including comprehensive developer API reference, tutorials, glossaries, and conceptual overviews. If you want to help other users by making a correction or addition, each page has a button at the bottom ([Edit this page on GitHub](#)) that takes you to the source file.

If you're reading this post, you should already know that Mapzen has a **blog** (<https://mapzen.com/blog/>). In our blog articles, you can commonly find information about product and event announcements, and applications of our tools, but also deep, engineering-related content. You can also sort posts by tags – **routing** (<https://mapzen.com/tag/routing>), **search** (<https://mapzen.com/tag/search>), **vector tiles** (<https://mapzen.com/tag/vector-tiles>), **tangram** (<https://mapzen.com/tag/tangram>), **transitland**

(<https://mapzen.com/tag/transitland>), **openstreetmap** (<https://mapzen.com/tag/osm>), **whosonfirst** (<https://mapzen.com/tag/whosonfirst/>) and **engineering** (<https://mapzen.com/tag/engineering/>) are just a few topics you can drill into.

We roll up our blog posts and other company news into a (mercifully short) monthly email newsletter. Sign up: <https://mapzen.com/newsletter-signup/> (<https://mapzen.com/newsletter-signup/>).

Email support

Mapzen's email address is **hello@mapzen.com** (<mailto:hello@mapzen.com>), which is monitored by engineers, product managers, and developer relations specialists. You can also find this address in the footer of every page on mapzen.com. Check in with us about anything you need, such as our APIs, workflow guidance, and rate limits. We use the questions you ask to improve the technical documentation so the answer is easier to find for the next person who has the same problem.

Please also send us links to your projects and tell us how you are using our tools in them. We want to see what you make with Mapzen!

Social media

We love hearing from you on our social media accounts. **Tweet us a hello** (<https://twitter.com/intent/tweet?text=@mapzen+hello+map+friends>), whether you have questions about our APIs or showing off your work. Follow **@mapzen** (<https://twitter.com/mapzen>) and you'll be the first to know about cool Mapzen things.

GitHub and Gitter

If you are working with the code for Mapzen's open-source projects, you might be more comfortable communicating with us through **GitHub** (<https://github.com>). GitHub is the collaborative version control system we use to manage our code and documentation. Posting an issue lets our engineers and other users know about a bug or software enhancement directly.

Gitter (<https://gitter.im>) is an instant message chat system for GitHub users. Some of Mapzen's development teams have set up chat rooms associated with our open-source projects so you can communicate with us and the community in real time.

- **Pelias** (https://gitter.im/pelias/pelias?utm_source=share-link&utm_medium=link&utm_campaign=share-link) for geographic search and geocoding
- **Tangram** (https://gitter.im/tangrams/tangram-chat?utm_source=share-link&utm_medium=link&utm_campaign=share-link) for the map-drawing engine
- **Tilezen** (https://gitter.im/tilezen/tilezen-chat?utm_source=share-link&utm_medium=link&utm_campaign=share-link) for vector tiles and map data

In-person support during office hours

If you are in the San Francisco area, Mapzen hosts open office hours every third Wednesday of the month. During office hours, you can get personalized technical assistance and training and meet other users. We provide bagels and light breakfast snacks, and lovely views of the San Francisco Bay from our office windows, plus video conference links to our staff elsewhere. Last month, we had simultaneous video calls between attendees and our engineers in New York and London!



The next event is Wednesday, September 21, 2016 from 9 to 11 A.M. If you sign up in advance, it helps us make sure we have the right staff available for your questions:

<http://goo.gl/forms/UdgrT7uLCocrSyNC2> (<http://goo.gl/forms/UdgrT7uLCocrSyNC2>)

· 31 August 2016 ·



Katie Kowalsky

Katie is a mapslainer who knows how to make a proper cheese plate and is a serial to-do list maker.



Rhonda Glennon

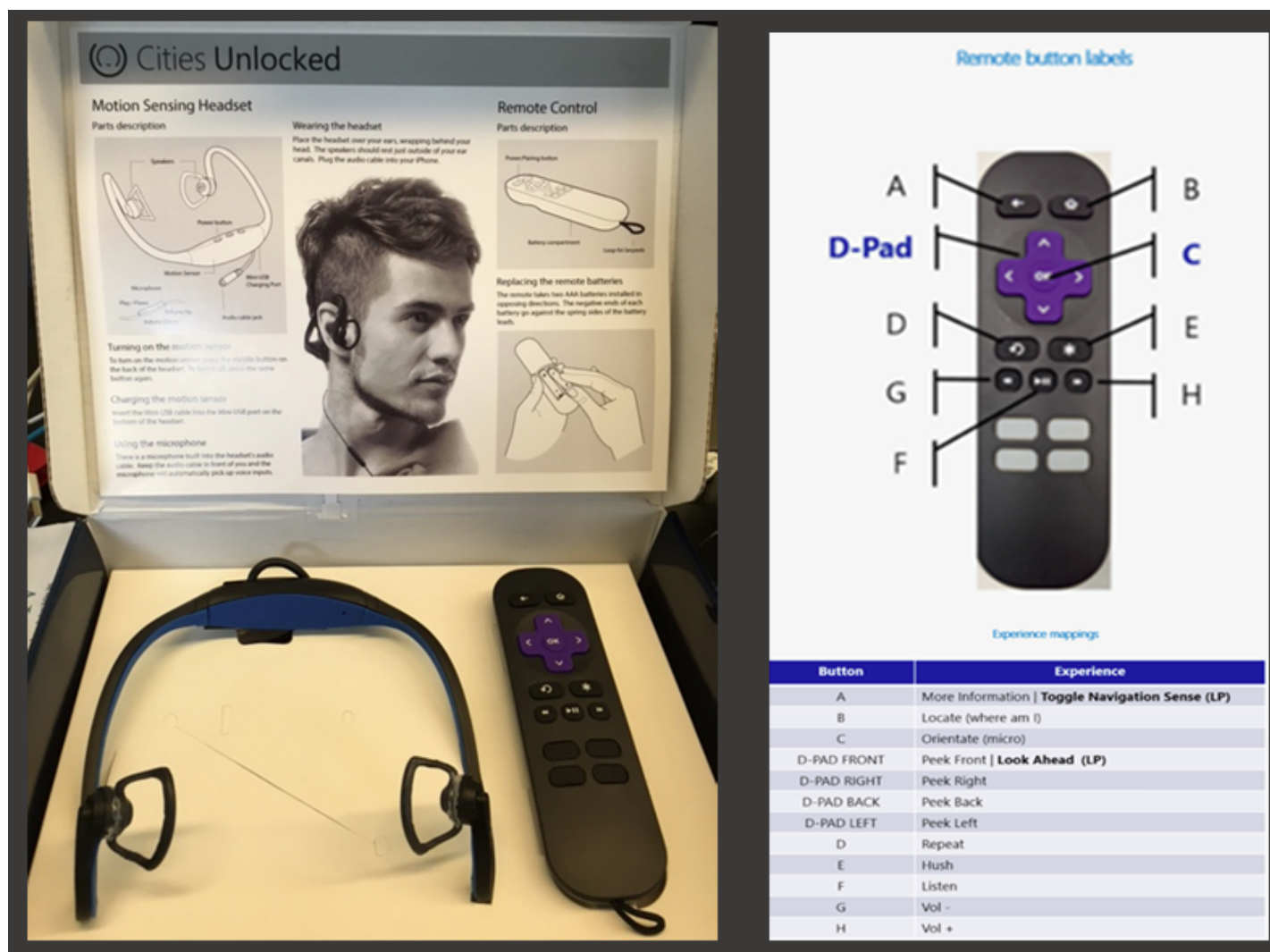
Rhonda is Mapzen's technical publications manager and writes about maps and developer tools.

© 2017 Mapzen

How Microsoft built an AR app with OpenStreetMap, ElasticSearch and Mapzen

mobility (/tag/mobility) vector-tiles (/tag/vector-tiles) data (/tag/data)

Erik Schlegel (<https://twitter.com/erikschlegel1>) dives into **how Microsoft built Cities Unlocked** (<https://www.microsoft.com/developerblog/real-life-code/2016/08/08/Querying-Spatial-Datasets-at-Scale-With-ElasticSearch.html>), an AR navigation system for visually impaired users.



Cities Unlocked announces location information throughout positional or head turn movements. The responsiveness and availability of our backend service had to be realtime. The planetary shapes we're working with are often comprised of complex geometry structures(i.e. parks and buildings). The expectation was our services can seamlessly onboard new cities to the platform, while maintaining acceptable performance for data volume growth.

Our mapping data provider would also have to have a strong story around accessibility(i.e. crossings, curbs, sidewalks, stairs, etc). The data model also had to be flexible so we can easily add new places, features and tags.

Microsoft turned to ElasticSearch, Mapzen vector tiles and Valhalla routing, and Wikipedia.

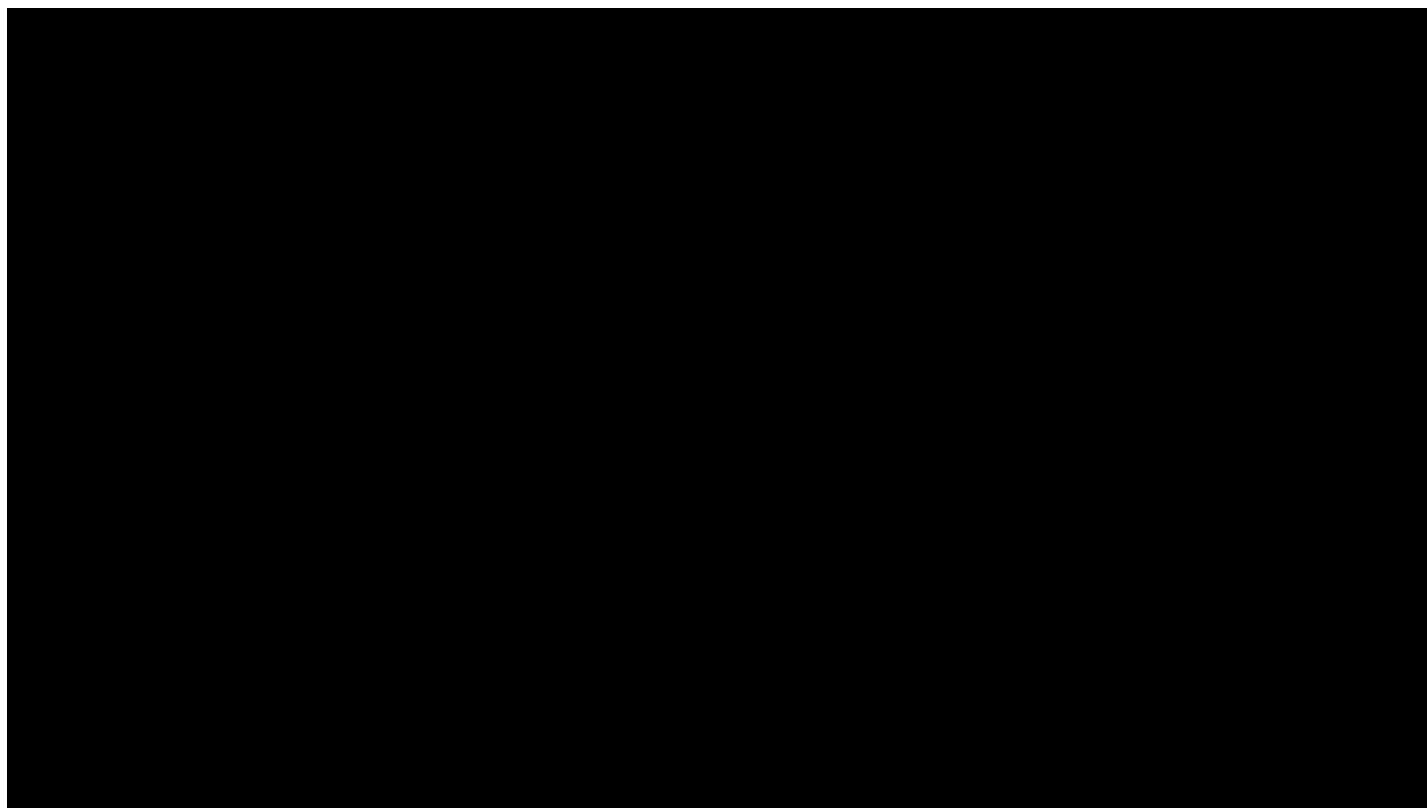
We implemented a vector tile based solution that collects mapping data as square shaped GeoTiles into ElasticSearch. Each tile is a square area of land with a defined size and location, identified by a row, height and zoom level(X/Y/Z). The highest zoom level comprises of 4 tiles. Each zoom level divides the parent tile into four separate tiles. We used a zoom level of 16(so an average bounding box of ~460 meters x ~460 meters)

We crowdsourced nearby geotile data based on the users position and a 3 kilometer radius distance. We're using Mapzens GeoJSON-based Vector Tile Service. Streets, buildings and parks are stored as GeoJSON LineString, while addresses and points of interest as a single GeoJSON point.

And best of all, they are giving data back to OpenStreetMap making it better for everyone.

Utilizing an open spatial data solution like OpenStreetMap was critical to our success, as we needed full read/write access to sidewalk objects(hydrants, mailboxes, trash cans, etc) to maintain an acceptable level of data quality. The current population of sidewalks and crossings in OSM are sparse. This presented a challenge as the open source Valhalla routing engine factors roads and intersections in the absence of pedestrian data in OSM. We're currently undergoing an exploratory effort that uses machine learning models paired with aerial and streetview imagery to infer sidewalk and crossing mapping features, with the goal of contributing that dataset back to OpenStreetMap to improve the data quality of Valhalla pedestrian routes.

If you want to learn more, Erik spoke spoke about this at **State of the Map US** (<http://stateofthemap.us/2016/osm-lights-up-the-world-for-blind-users-with-sound/>) in July:



We are thrilled to involved with this effort as it falls in line with **our philosophy on mobility and accessibility** (<https://mapzen.com/blog/introducing-mapzen-mobility/>) where people and the freedom of movement come first.



Mapzen

Opening maps with open source and open data.

© 2017 Mapzen

A Routing Engine Of One's Own: DIY Valhalla

demo (/tag/demo) **routing** (/tag/routing) **engineering** (/tag/engineering)

Mapzen's software has always been open source, including **Valhalla, our powerful routing and navigation engine** (<https://mapzen.com/products/turn-by-turn/>) that powers our Turn-by-Turn service. Anyone can download the code and run it for themselves, but the compilation step can be a barrier to many.

Starting with Valhalla, we are making parts of our server-side code available via **Ubuntu**

Personal Package Archives

(https://help.ubuntu.com/community/Repositories/Ubuntu#Adding_Personal_Package_Archives_.28PPAs.29) or PPAs. This makes it possible to install with a simple `apt-get` command, which in turn makes it easy to include Valhalla on a fresh Ubuntu server or as part of a container recipe for Docker or Vagrant. **We've published Valhalla to** `ppa:mapzen/routing` (<https://launchpad.net/%7Emapzen/+archive/ubuntu/routing>).

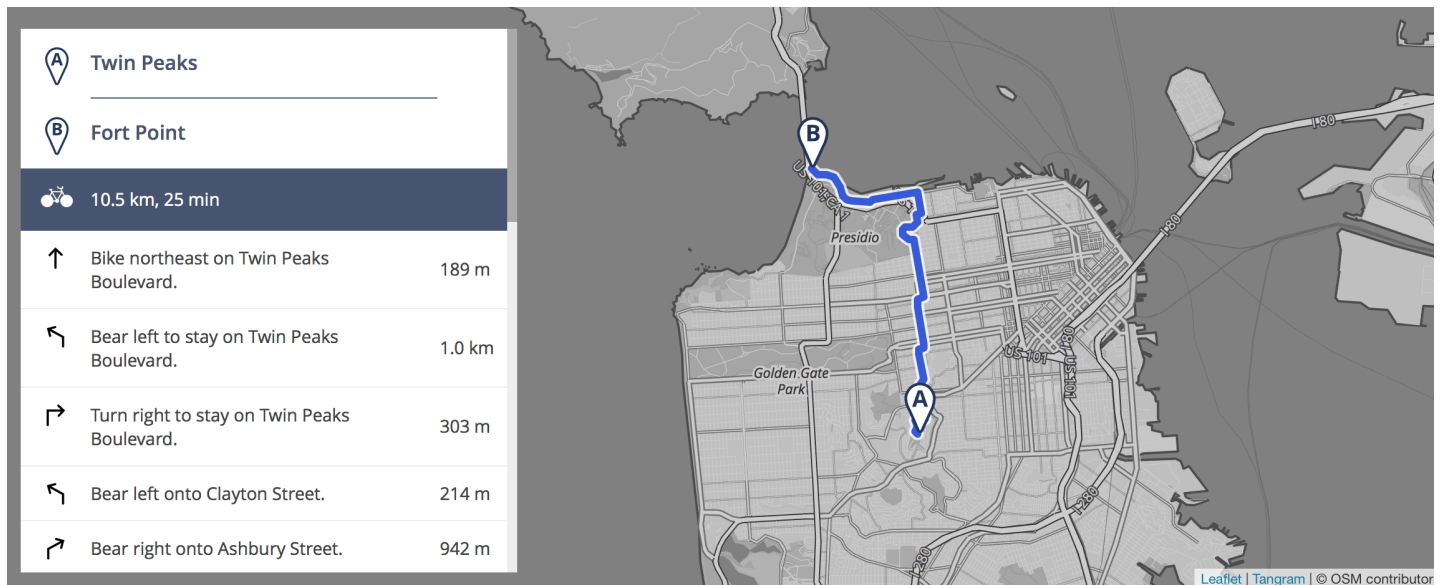
Whether you want to experiment or need a very large number of routing requests for analysis, now you can get Valhalla routing working on your own server in just a few minutes!

```
# Add Mapzen routing PPA
add-apt-repository -y ppa:mapzen/routing
apt-get update -y

# Install Valhalla code
apt-get install -y valhalla-server

# Try a multimodal route in San Francisco
curl -g 'localhost:8002/route?json={"locations":[{"lat":37.764472,"lon":-122.422027},{"la
```

That's it. Valhalla comes with data for San Francisco, and a simple interactive demo available at your `http://localhost/demo.html` (<http://localhost/demo.html>):



We've also made data available for all of the SF Bay Area, Rome and New York City in one-time sample files:

- **sf-bay-area-2016-09-07.tgz** (<https://mapzen-assets.s3.amazonaws.com/images/valhalla-package/sf-bay-area-2016-09-07.tgz>) (84MB)
- **rome-2016-09-07.tgz** (<https://mapzen-assets.s3.amazonaws.com/images/valhalla-package/rome-2016-09-07.tgz>) (52MB)
- **new-york-2016-09-07.tgz** (<https://mapzen-assets.s3.amazonaws.com/images/valhalla-package/new-york-2016-09-07.tgz>) (101MB)

You can install the above data packages just by unpacking them into the directory `/var/valhalla-server` :

```
# Unpack new tiles into /var/valhalla-server
tar -C /var/valhalla-server -xzvf rome-2016-09-07.tgz

# Restart Valhalla to use the new tile data
/etc/init.d/valhalla-server restart
```

Notes:

- We're working on how to best deliver data for other cities, stay tuned for more.
- If you are testing multimodal routing, note that these data packages include transit data for SF, NYC and Rome for September 2016, so be sure to include relevant data ranges in your URL if you aren't running this in September 2016. Details on `use_` and `date` options

are discussed in the **transit routing blog post** (<https://mapzen.com/blog/transit-routing>).

- For reference, here is a sample multimodal route that specifies dates and transit weights:

```
localhost:8002/route?json={"costing_options":{"transit":{"use_bus":0.5,"use_rail":0.
```

- See **Mapzen documentation** (<https://mapzen.com/documentation/mobility/turn-by-turn/api-reference/>) for more useful information on Valhalla.
- Limit for the number of stops and distance for can be changed in `/etc/valhalla-server.json` (note that longer requests with more stops will use more memory!)
- **Time-Distance Matrix** (<https://mapzen.com/documentation/mobility/matrix/>) and **Optimized route requests** (<https://mapzen.com/documentation/mobility/optimized/>) are not yet supported in this package.

Whether you route by car, foot, bike or transit, **let us know what you think** (<mailto:valhalla@mapzen.com?subject=Valhalla%20PPA%20FTW>) – suggestions are welcome!

· 08 September 2016 ·



Michal Migurski

Oakland-dwelling geodata cyclist.

© 2017 Mapzen

Tangram Heightmapper

tangram (/tag/tangram) **terrain-tiles** (/tag/terrain-tiles) **terrain** (/tag/terrain)

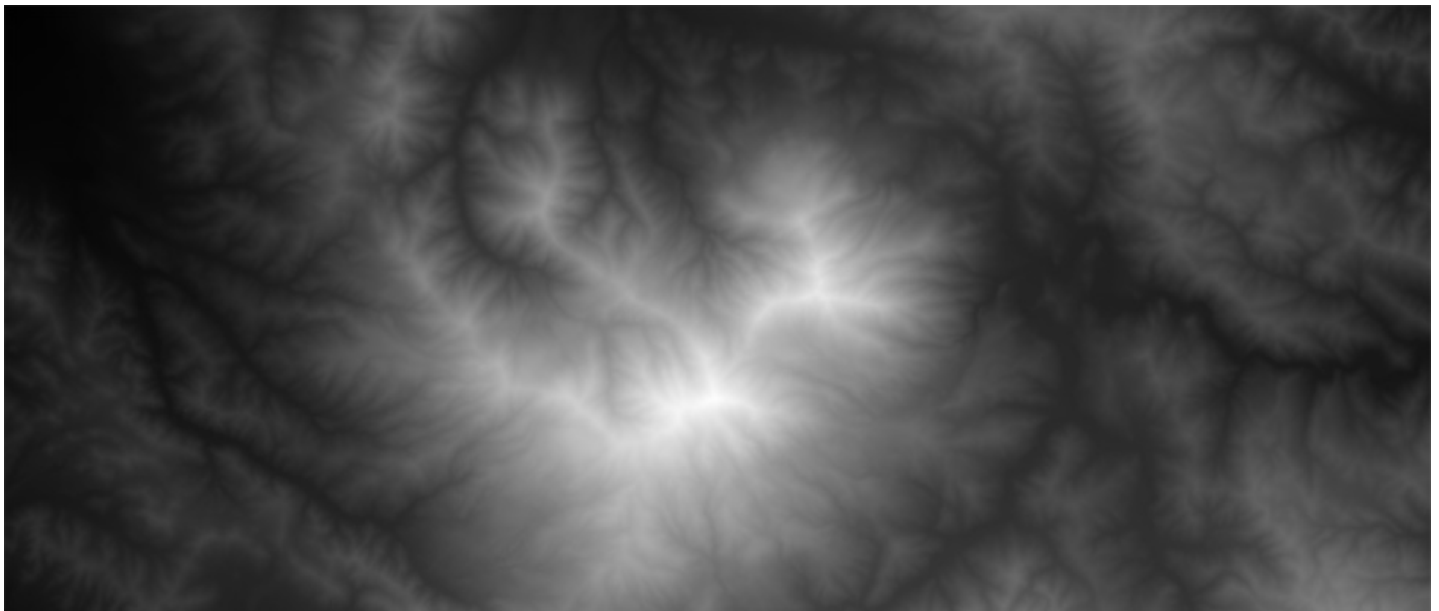
Mapzen **recently released** (<https://mapzen.com/blog/elevation/>) a whole lot of high-quality open-source terrain data in the form of a pair of global tilesets, using raster images to store the elevation and slope of the surface of the earth.


I've previously described **some ways to render this data** (<https://mapzen.com/blog/mapping-mountains>) in a 2D context, but it can also be used to generate 3D models, like this one of Mt. Diablo in California:



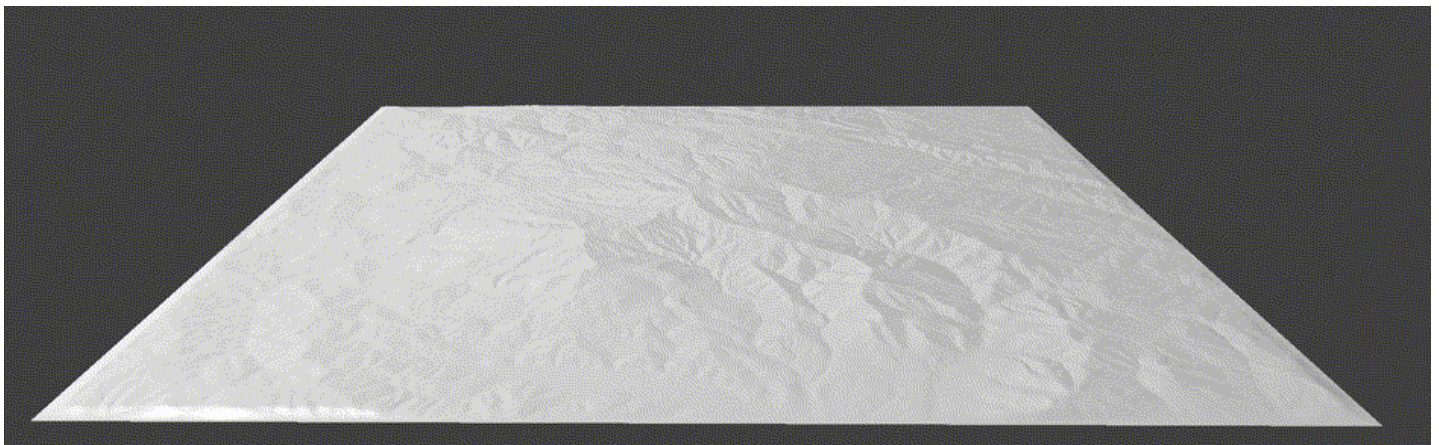
I made a viewer called **Heightmapper** (<https://tangrams.github.io/heightmapper>) to browse the terrain tiles and create heightmaps from them, which are grayscale images of terrain data.

Here's the above view in the Heightmapper, roughly (the printed model is south-side-up):



(<https://tangrams.github.io/heightmapper/#13.58529/37.8835/-121.9163>) (*Open full screen*  (<https://tangrams.github.io/heightmapper/#13.58529/37.8835/-121.9163>))

A heightmap like this one can be turned into a 3D model by using it as a “**displacement map** (https://en.wikipedia.org/wiki/Displacement_mapping)”, which displaces the vertices of a 3D mesh according to the brightness of each pixel, like this:



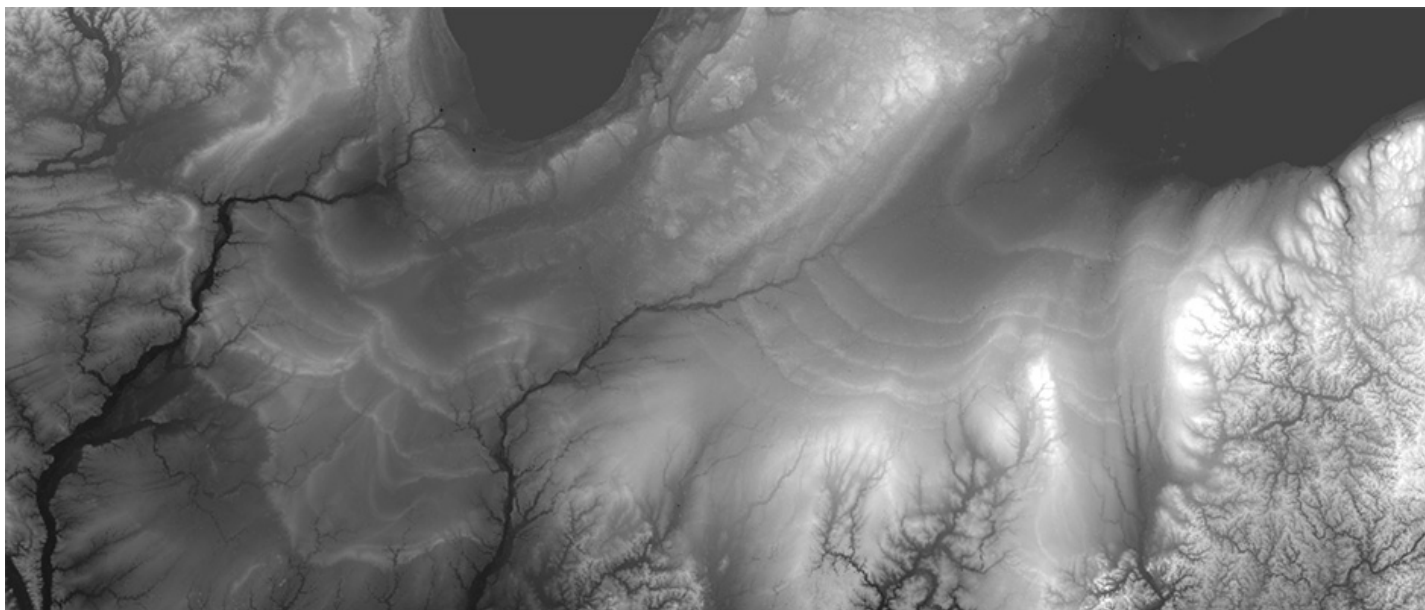
By default, Heightmapper is in “auto-exposure” mode, to ensure that the current view’s elevation values are spread out over every gray in the grayscale. This mode checks the highest and lowest values in the current view, and sets the levels to make use of the full tonal range.

To aid in scaling the resulting models, a “scale factor” in Heightmapper’s control panel shows the vertical range of the current view, expressed as a ratio in terms of the view’s width. When you export, use this factor to determine how tall your model should be on the z-axis. (I made **a brief tutorial for doing this in Blender** (https://github.com/tangrams/heightmapper/blob/gh-pages/exporting_to_blender.md), a popular but confusing open-source 3D application.)

Blame it on Terrain

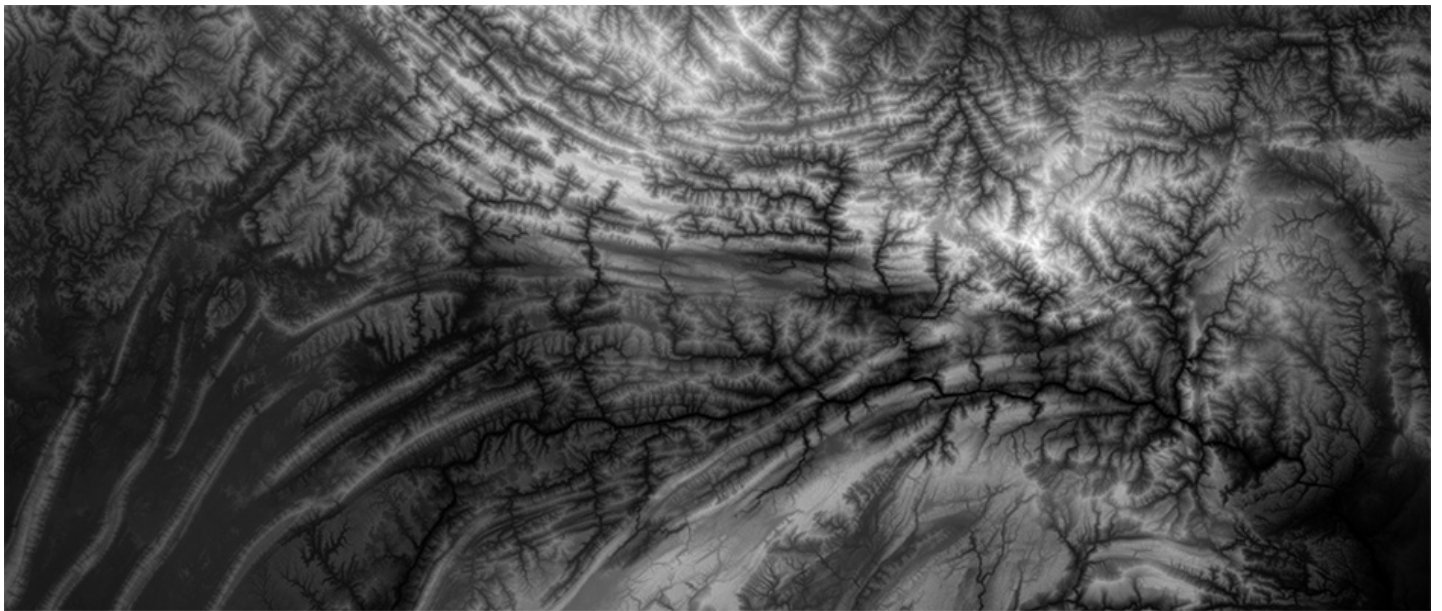
This view also reveals surface details which aren't easily visible even in a 3D model. By turning up the contrast on the data, it provides glimpses into the deep history of the landscape, hidden in small differences in the terrain. For example:

The prehistoric coastlines of the **Great Lakes**
(https://en.wikipedia.org/wiki/Great_Lakes#Geology):



(<https://tangrams.github.io/heightmapper/#7.65371/41.215/-86.154>) (*Open full screen ↗*
(<https://tangrams.github.io/heightmapper/#7.65371/41.215/-86.154>))

The folded mountains which give shape to the **Three Gorges**
(https://en.wikipedia.org/wiki/Three_Gorges):



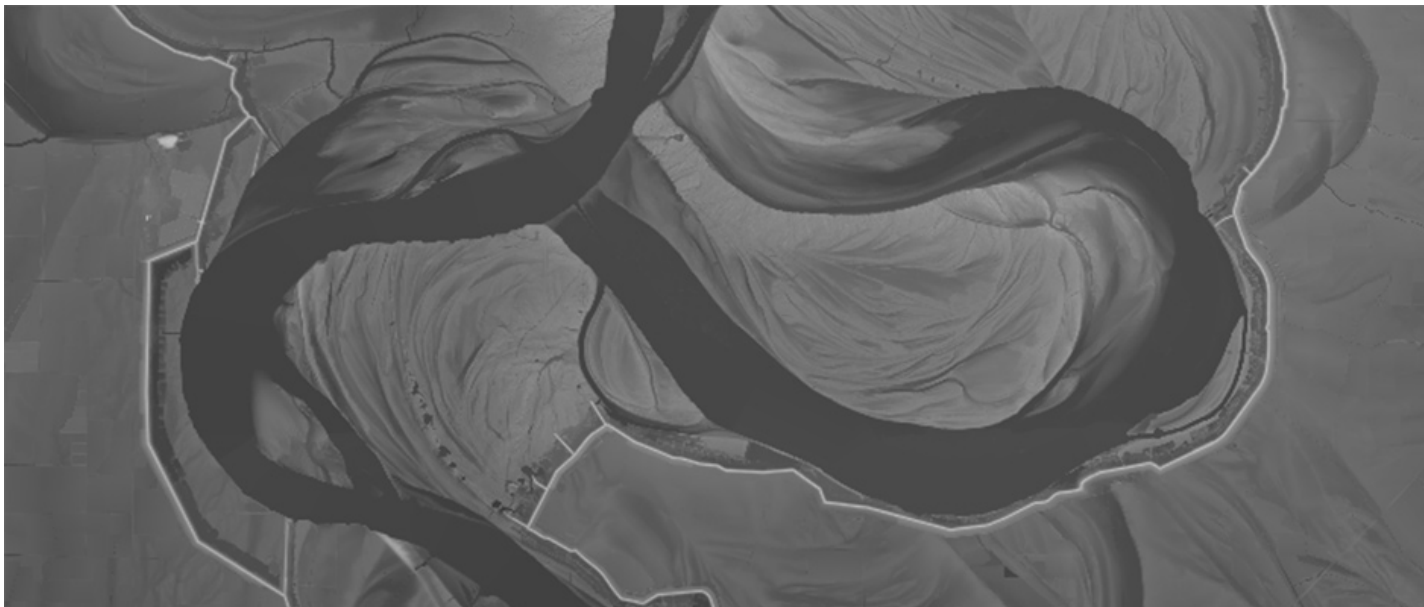
(<https://tangrams.github.io/heightmapper/#8.49583/31.4034/109.0664>) (*Open full screen ↗*
(<https://tangrams.github.io/heightmapper/#8.49583/31.4034/109.0664>))

The fractal boundary between watersheds in central India:



(<https://tangrams.github.io/heightmapper/#9/18.7737/75.6450>) (*Open full screen ↗*
(<https://tangrams.github.io/heightmapper/#9/18.7737/75.6450>))

The overlapping former paths of the Mississippi, now bounded by levees:



(<https://tangrams.github.io/heightmapper/#12.02403/33.6859/-91.1265>) (*Open full screen ↗*
(<https://tangrams.github.io/heightmapper/#12.02403/33.6859/-91.1265>))

Check out the Heightmapper code or fork it at the **github repo**
(<http://github.com/tangrams/heightmapper>).

· 13 September 2016 ·



Peter Richardson

Peter vexes vertices, flusters fragments, and pesters pixels on Mapzen's graphics team.

© 2017 Mapzen

Announcing Unified API Keys

geocoding (/tag/geocoding) **routing** (/tag/routing) **search** (/tag/search)

vector-tiles (/tag/vector-tiles)



The Keys of Westphalia (https://en.wikipedia.org/wiki/Peace_of_Westphalia)

We've made **Mapzen API keys** (<https://mapzen.com/documentation/overview/#developer-accounts-and-api-keys>) simpler and easier to use this week.

If you've used our services in the past, you might recall having to create separate API keys for each service: a key for **Search** (<https://mapzen.com/products/search/>), a key for **Turn-by-Turn** (<https://mapzen.com/products/turn-by-turn/>), and another for **Vector Tiles** (<https://mapzen.com/projects/vector-tiles>). For a complex project, you might have had to juggle as many as a half-dozen keys. Starting today, we're unifying our API keys across services so you just need a single key for a project. Any API key will work for all Mapzen services.

In addition, your existing keys now grant access to all services, so you won't need to go back and change your work.

This change should be completely transparent to you. Each service still has separate rate limits. If you've **talked to us about your project** (<mailto:hello@mapzen.com>) and we've customized your **API rate limits** (<https://mapzen.com/documentation/overview/#rate-limits>), you will still be able to use the custom rate limits we've set. Head to **mapzen.com/developers** (<https://mapzen.com/developers>) to get started using our services!

Note: This post was updated on 20 September 2016 to clarify that existing keys work for all services, too.

· 16 September 2016 ·



Michal Migurski

Oakland-dwelling geodata cyclist.

© 2017 Mapzen

Customize Mapzen Turn-by-Turn narrative language for all ports

routing (/tag/routing) **demo** (/tag/demo)

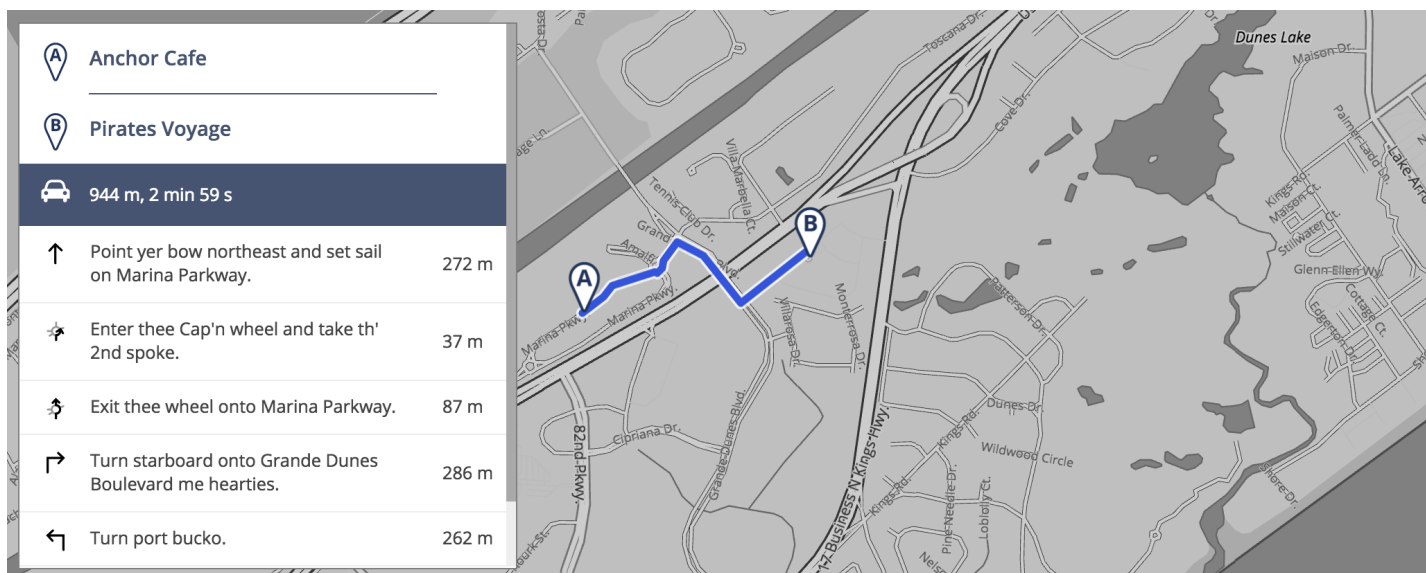
Ahoy, me hearties - it is time for ye landlubbers to charter a course for thee Cap'n! And ye best be taking the fastest route or ye be walkin the plank! What's the matter, you scallywag? Have you never heard Pirate before? Well, blimey! In honor of **International Talk Like A Pirate Day** (<http://talklikeapirate.com/>), Mapzen is highlighting this **Pirate narrative language** (<https://github.com/valhalla/odin/blob/master/locales/en-US-x-pirate.json>) with the hopes that this will act as the catalyst that will encourage the contribution of other languages for **Mapzen Turn-by-Turn** (<https://mapzen.com/products/turn-by-turn>).

While a few of ye scurvy dogs have been busy swabbing the deck, some of the savvy buccaneers have been busy contributing languages so that the Mapzen Turn-by-Turn service may be used in all ports. We would save ye the finest rum as a token of our appreciation, but as ye know, the rum is always gone. Therefore, a thank you will have to suffice.

Currently, we support the following languages:

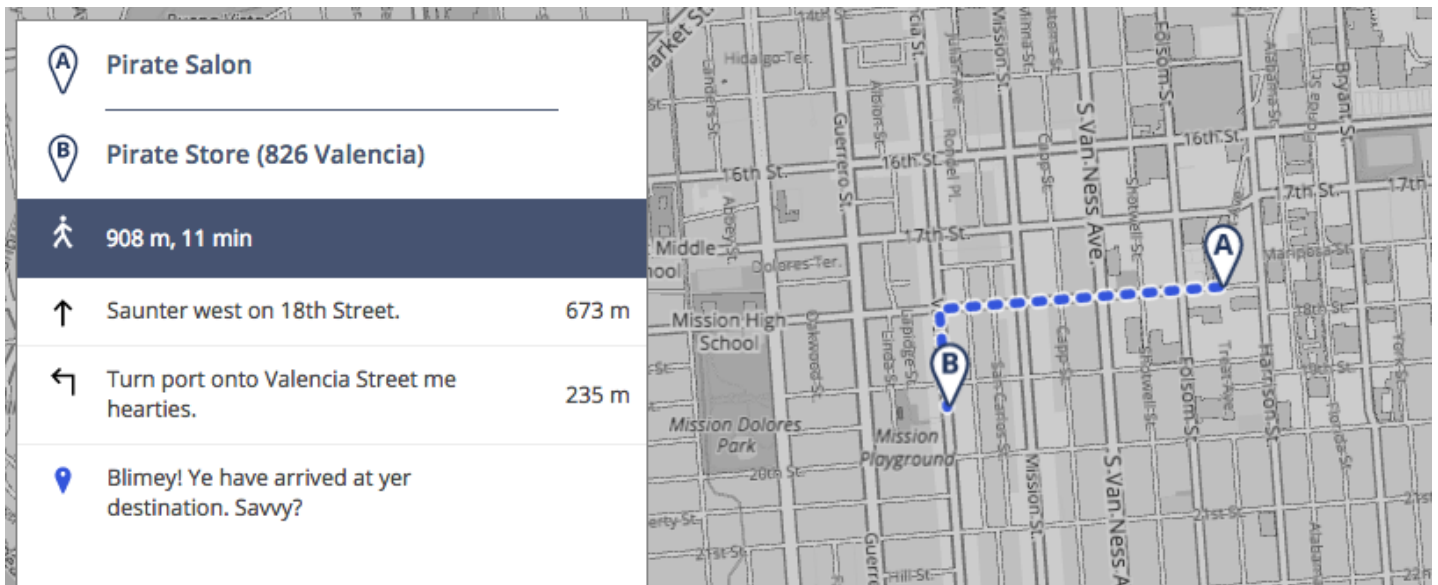
- Czech
- English(US)
- French
- German
- Italian
- Pirate
- Spanish

Look at the treasure map below, me scurvy dogs, and find a demo that is using the Pirate narrative language! **Set sail on a voyage to the Carolina coast** (<https://mapzen.com/resources/projects/turn-by-turn/language/pirate.html?d=0>)!



(<https://mapzen.com/resources/projects/turn-by-turn/language/pirate.html?d=0>)

Then **tack hard out of the cove** (<https://mapzen.com/resources/projects/turn-by-turn/language/pirate.html?d=1>) to find Treasure Island. And with your booty, **navigate to the Pirate Store** (<https://mapzen.com/resources/projects/turn-by-turn/language/pirate.html?d=2>) barter for supplies after getting your hair curled!



(<https://mapzen.com/resources/projects/turn-by-turn/language/pirate.html?d=2>)



(<https://visitoffice.com/work/826-valencia/>)

And to help you buckos achieve success, the first mate has provided a **link** (<https://github.com/valhalla/odin/tree/master/locales>) to the repository where the languages are stored. And shiver me timbers - there are also instructions on how to contribute your own language, which are detailed below:

- Copy the **en-US.json** (<https://github.com/valhalla/odin/blob/master/locales/en-US.json>) to `<NEW_LANGUAGE_TAG>.json` Using **IETF BCP 47** (<https://tools.ietf.org/html/bcp47>) as reference - the typical format for the `<NEW_LANGUAGE_TAG>` is:

<ISO 639 two-letter language code (https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes)>-<ISO 3166 two-letter country code (https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2)>

Czech/Czech Republic example: `cs-CZ`

- Update the `posix_locale` value in your new file. The character encoding must be UTF-8. The typical format is:

<ISO 639 two-letter language code (https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes)>_<ISO 3166 two-letter country code (https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2)>.UTF-8

UTF-8 Czech/Czech Republic `posix_locale` example: `cs_CZ.UTF-8`

- Update the `aliases` array in your new file. A typical alias entry is the **ISO 639 two-letter language code (https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes)** without the **ISO 3166 two-letter country code (https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2)**. The alias entry must be unique across language files.

Czech `aliases` entry example: `cs`

- Do not translate the JSON keys or phrase tags. An example using the ramp instruction:

The **JSON keys** and the **phrase tags** are highlighted in **red** below
Please **do not** translate these items

```
"ramp": {
  "phrases": {
    "0": "Take the ramp on the <RELATIVE_DIRECTION>.",
    "1": "Take the <BRANCH_SIGN> ramp on the <RELATIVE_DIRECTION>.",
    "2": "Take the ramp on the <RELATIVE_DIRECTION> toward <TOWARD_SIGN>.",
    "3": "Take the <BRANCH_SIGN> ramp on the <RELATIVE_DIRECTION> toward <TOWARD_SIGN>.",
    "4": "Take the <NAME_SIGN> ramp on the <RELATIVE_DIRECTION>.",
    "5": "Turn <RELATIVE_DIRECTION> to take the ramp.",
    "6": "Turn <RELATIVE_DIRECTION> to take the <BRANCH_SIGN> ramp.",
    "7": "Turn <RELATIVE_DIRECTION> to take the ramp toward <TOWARD_SIGN>.",
    "8": "Turn <RELATIVE_DIRECTION> to take the <BRANCH_SIGN> ramp toward <TOWARD_SIGN>.",
    "9": "Turn <RELATIVE_DIRECTION> to take the <NAME_SIGN> ramp."
  },
  "relative_directions": ["left", "right"],
  "example_phrases": {
    "0": ["Take the ramp on the left.", "Take the ramp on the right."],
    "1": ["Take the I 95 ramp on the right."],
    "2": ["Take the ramp on the left toward JFK."],
    "3": ["Take the South Conduit Avenue ramp on the left toward JFK."],
    "4": ["Take the Gettysburg Pike ramp on the right."],
    "5": ["Turn left to take the ramp.", "Turn right to take the ramp."],
    "6": ["Turn left to take the PA 283 West ramp."],
    "7": ["Turn left to take the ramp toward Harrisburg/Harrisburg International Airport."],
    "8": ["Turn left to take the PA 283 West ramp toward Harrisburg/Harrisburg International Airport."],
    "9": ["Turn right to take the Gettysburg Pike ramp."]
  }
},
```

- Please translate the JSON values. As needed, reorder the phrase words and tags - the tags must remain in the phrase. An example using the ramp instruction:

The **JSON values** (excluding the **phrase tags**) are highlighted in **green** below
Please translate these items

```
"ramp": {
  "phrases": {
    "0": "Take the ramp on the <RELATIVE_DIRECTION>.",
    "1": "Take the <BRANCH_SIGN> ramp on the <RELATIVE_DIRECTION>.",
    "2": "Take the ramp on the <RELATIVE_DIRECTION> toward <TOWARD_SIGN>.",
    "3": "Take the <BRANCH_SIGN> ramp on the <RELATIVE_DIRECTION> toward <TOWARD_SIGN>.",
    "4": "Take the <NAME_SIGN> ramp on the <RELATIVE_DIRECTION>.",
    "5": "Turn <RELATIVE_DIRECTION> to take the ramp.",
    "6": "Turn <RELATIVE_DIRECTION> to take the <BRANCH_SIGN> ramp.",
    "7": "Turn <RELATIVE_DIRECTION> to take the ramp toward <TOWARD_SIGN>.",
    "8": "Turn <RELATIVE_DIRECTION> to take the <BRANCH_SIGN> ramp toward <TOWARD_SIGN>.",
    "9": "Turn <RELATIVE_DIRECTION> to take the <NAME_SIGN> ramp."
  },
  "relative_directions": ["left", "right"],
  "example_phrases": {
    "0": ["Take the ramp on the left.", "Take the ramp on the right."],
    "1": ["Take the I 95 ramp on the right."],
    "2": ["Take the ramp on the left toward JFK."],
    "3": ["Take the South Conduit Avenue ramp on the left toward JFK."],
    "4": ["Take the Gettysburg Pike ramp on the right."],
    "5": ["Turn left to take the ramp.", "Turn right to take the ramp."],
    "6": ["Turn left to take the PA 283 West ramp."],
    "7": ["Turn left to take the ramp toward Harrisburg/Harrisburg International Airport."],
    "8": ["Turn left to take the PA 283 West ramp toward Harrisburg/Harrisburg International Airport."],
    "9": ["Turn right to take the Gettysburg Pike ramp."]
  }
},
```

- Run `make check` on the **odin** (<https://github.com/valhalla/odin>) repo to verify the tests pass OR move on to the last step and we can help verify.
- Submit a pull request for review. Thank you!

Ahoy, ye mateys! Alas, yer destination is up ahead! Every pirate knows two things: First, ye set sail on an adventure to seek yer treasure. Secondly, dead men tell no tales. At Mapzen, we know two things: First, we set sail on an adventure to provide a savvy open service. Secondly, it is wise mateys that tell their tales and contribute their native narrative language.

Check out the **Mapzen Turn-by-Turn documentation** (<https://mapzen.com/documentation/turn-by-turn/>) and **let us know if you have any questions** (<https://twitter.com/intent/tweet?text=@Mapzen%20@ValhallaRouting%20Hi!>).

· 19 September 2016 ·



Duane Gearhart

Duane is a software engineer @mapzen specializing in quality route guidance and real-world route analysis.



John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

Mapping from the Brussels Waffle House

osm (/tag/osm)

Mapzen has made landfall in the land of waffles, beer and french fries (aka frites) to further support two of the most important mapping conferences of the year. No, we didn't join the 27 heads of government at the European Commission State of the Union address (at least not this year). We were busy. But we ARE here and sponsoring the **Humanitarian OpenStreetMap Team (HOT) Summit** (<http://summit.hotosm.org/>) on Thursday, September 22nd and the **International State of the Map (SOTM)** (<http://2016.stateofthemap.org>) conference on Friday through Monday, September 23rd-26th.

Come find us!! We want to hear from you!! We also want to give you the **latest Null Island T-shirt** (<https://twitter.com/mapzen/status/756909927973605376>)!!



We'll have a table at SOTM with people to answer your questions, **stickers** (<https://mapzen.com/blog/sotmus-seattle-update/>) to decorate your computer, and null island t-shirts to complete your mapping wardrobe. Limited supply only (aka what we could

carry on a plane).

Julian will be doing a lightning talk about the history of Denmark and Sweden's power struggle as captured by a geocoding mistake. **Check twitter (<https://twitter.com/mapzen>)** for the time of the talk! **BORNHOLM (<https://mapzen.com/blog/assult-on-copenhagen/>)!**

And I, Alyssa, will be joining all the OpenStreetMap US board members to talk about OpenStreetMap and the real deal of getting shit done. Also called "**The Real McCoy (<http://2016.stateofthemap.org/2016/openstreetmap-us-the-real-mccoy>)**," you can find it all on Saturday at 3pm, Auditorium A. The Real McCoy is talking the approach to community events, communication strategies, innovation education and (get this) MONEY. OpenStreetMap US is one of the largest, most successful local open source communities. Learn how it works.

Open geo data is the foundation of ALL of Mapzen's services. If **you're in Brussels (<https://alloftheplaces.xyz/1042196595>)**, come to our table and learn what that means for your work. And if you're afar, **contact us to learn more (<mailto:hello@mapzen.com?subject=hallo>)**. Or **tweet with an emoji (<https://twitter.com/intent/tweet?text=%F0%9F%87%A7%F0%9F%87%AA%F0%9F%8D%9F>)**. We like that too.

*preview image via **Nathan Gray (<https://www.flickr.com/photos/a-culinary-photo-journal/6234480616>)***

· 22 September 2016 ·



Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.

© 2017 Mapzen

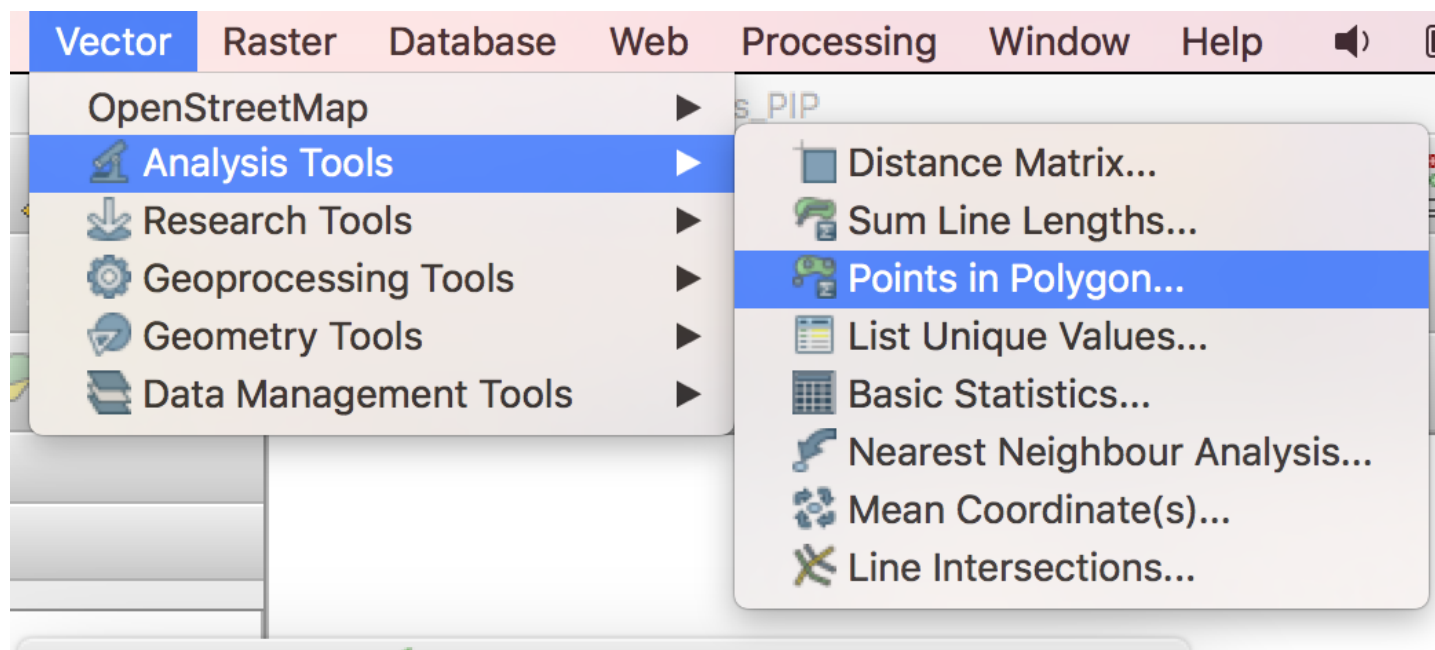
Pools and Polls -- Coloring Choropleths

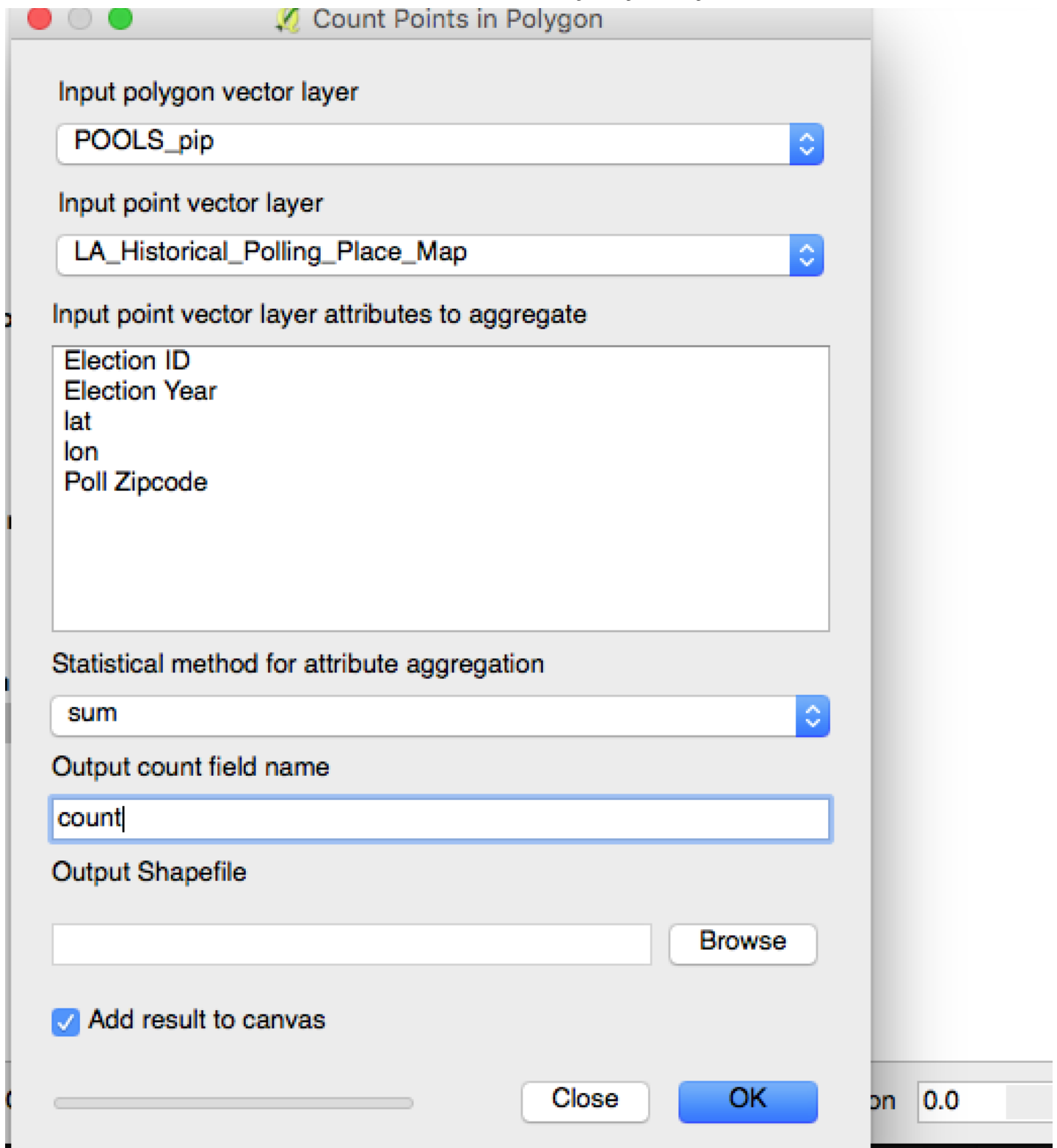
tangram (/tag/tangram) **data** (/tag/data) **demo** (/tag/demo)

Backyard pools are a foreign concept to those of us in San Francisco. As Karl the Fog rolls up, around and down Twin Peaks, the last place most people want to be is in an open body of water. While we take solace in our burrito superiority, we leave swimming in your backyard to the people of Los Angeles **and their 250,000 pools** (<http://thethrust.net/diving-into-las-pools/>).

But are these pools on the map? Sort of. The **L.A. County Assessor's office** (<https://data.lacounty.gov/Parcel-/Assessor-Parcels-Data-2006-thru-2015/9trm-uz8i/about>) has data on single family homes with pools, but not the shapes of the pools themselves. Meanwhile, the OpenStreetMap community of Southern California has started a **MapRoulette challenge** (<http://maproulette.org/map/494/639209>) to import more of them – only 245,000 to go!

I was curious to see where this pool mapping activity was happening and decided to make a choropleth map. I downloaded the **pools traced to date in OSM** (<http://overpass-turbo.eu/s/imW>) along with LA neighborhood boundaries as **GeoJSON** (<http://boundaries.latimes.com/set/la-county-neighborhoods-v6/>) from the **Los Angeles Times boundary API** (<http://boundaries.latimes.com/about/>). Then I fired up QGIS and ran Points in Polygon:





Count Points in Polygon

Input polygon vector layer

POOLS_pip

Input point vector layer

LA_Historical_Polling_Place_Map

Input point vector layer attributes to aggregate

- Election ID
- Election Year
- lat
- lon
- Poll Zipcode

Statistical method for attribute aggregation

sum

Output count field name

count

Output Shapefile

Browse

☒ Add result to canvas

Close OK

I then told it to look for the new 'count' value, and picked a graduated color ramp:

General

Column: 123 count

Symbol: Change...

Legend Format: %1 - %2 Precision: 1 ☐ Trim

Method: Color

Color ramp: [source] ☐ Invert

Classes **Histogram**

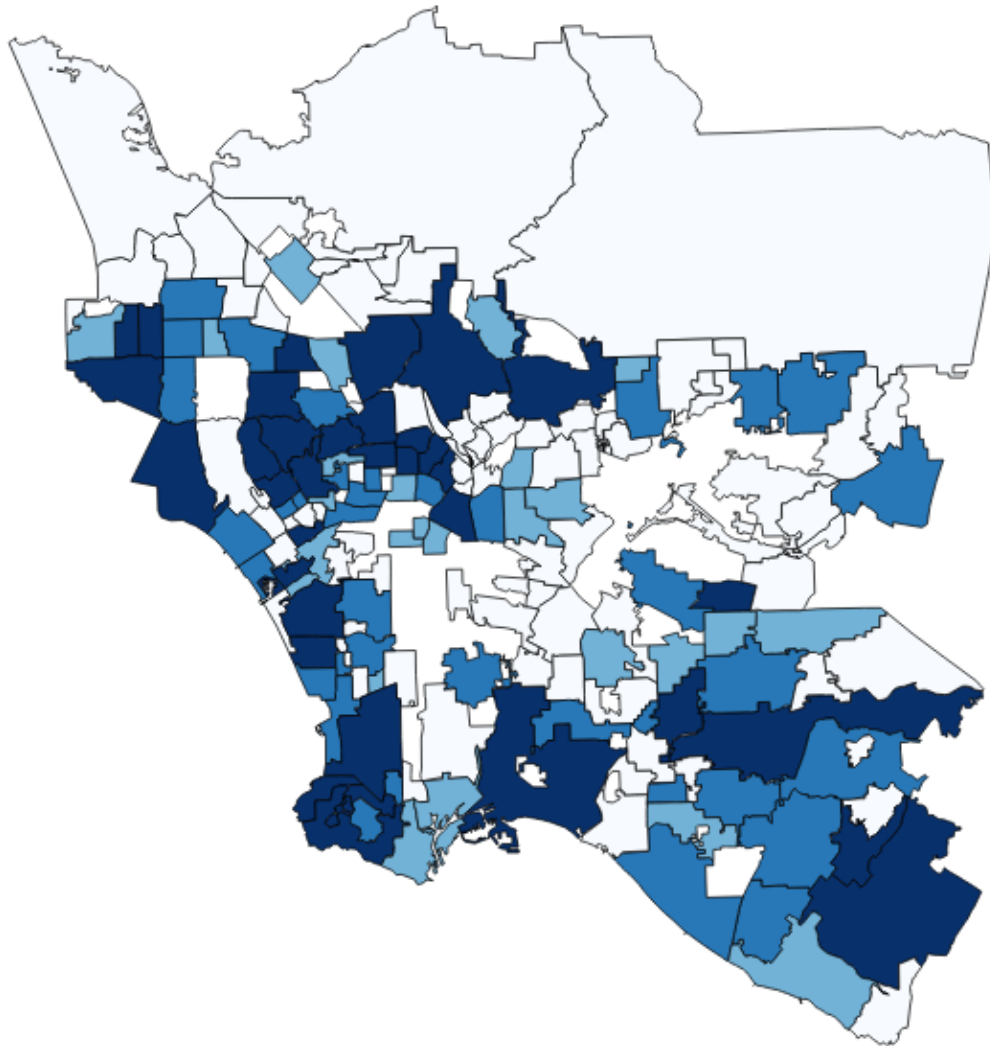
Mode: Quantile (Equal Count) Classes: 5

Symbol	Values	Legend
<input checked="" type="checkbox"/>	0.000 - 1.000	0.0000 - 1.0000
<input checked="" type="checkbox"/>	1.000 - 1.000	1.0000 - 1.0000
<input checked="" type="checkbox"/>	1.000 - 3.200	1.0000 - 3.2000

☒ Link class boundaries

Layer rendering

and got a decent looking map of OSM activity in Los Angeles County!



But how to share this “Pools in Polygons” map with the good people of Los Angeles? Exporting the newly PIP’d neighborhoods as GeoJSON, I uploaded the data to gist and brought it into Tangram:



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

Los Angeles OSM Pool Import by Neighborhood

Pools in Polygons (https://tangrams.github.io/tangram-frame/?lib=0.10&url=https://mapzen-assets.s3.amazonaws.com/resources/coloring-choropleths/la_pools_choropleth.yaml#10/33.9587/-118.1621)

You can take a look at the source YAML for the map **in Tangram Play** (<https://mapzen.com/tangram/play/?lines=40-51&scene=https%3A%2F%2Fmapzen-uploads.s3.amazonaws.com%2Fplay%2F262792e74b86d8cb772a3f0fd83c4328%2Fla-pools%2Fscene.yaml#9.7769/34.0081/-118.2312>).

I imported Refill as a base layer and set up some `styles` to make sure Refill showed up through the choropleths. I then imported the PIP'd neighborhood GeoJSON along with the points for the pools themselves.

`layers` is where the action is. I defined the size, color and visibility of the pool points:

```

layers:
  pools_of_interest_z10:
    data: { source: pools_of_interest}
    filter: { $zoom: { min: 10, max: 18} }
    draw:
      points:
        color: [1,1,0,0.7]
        size: [[10,4px],[18,14px]]
        order: 500

```

(Note how `size: [[10,4px],[18,14px]]` dynamically resizes points as you move in from `z10` to `z18`.)

I then drew the neighborhoods and colored them according to the value for `count` :

```

neighborhoods_pools:
  data: { source: polygons_in_pools}
  filter: function() { return feature.count > 0; }
  borders:
    draw:
      lines:
        order: 500
        color: aqua
        width: [[8,0.1px],[18,4px]]
      choropleth:
        order: 500
        color: |
          function() {
            return "hsl(" + (200 + (feature.count/500 * 55)) + ",100%,55%)";
          }

```

You can define colors many different ways in Tangram – named colors, RGB, hex, or HSL. I chose the latter, and using an inline Javascript function, generated a range of colors of blue starting at **hsl(200)** (<http://hslpicker.com/#1ab2ff>), scaling it based on the value of `count`. This took some trial and error but I was able to generate an attractive and reasonable range of colors for the choropleth.

Neighborhoods appear first, then dots come in at `z10` to give an indication of the distribution of pools in each neighborhood. Neighborhood names come in at `z11` , and the pools themselves are outlined starting at `z18` .

The OSM pool Maproulette is interesting but the imported data (never mind all of the pools) are unevenly distributed, and many neighborhoods have no data at all. I wanted to try making a choropleth out of a more evenly distributed open data set.

I had high hopes for mapping taco trucks, because `tacos`. I grew excited (and hungry) when I discovered that the LA County Department of Health has health inspection scores of over 3000 food trucks, nearly 400 of which had `taco` in the name. But after geocoding their addresses using a Mapzen Search plugin for Google Sheets that we are testing, I realized that they were even more clustered than pools! It turns out the dataset only had the addresses of where the trucks slept at night, not where they delivered their delicious taco payload during the day.

This made me hungrier so I got a burrito. While savoring my al pastor and perusing the LA County Open Data portal, I came across a dataset of polling places. (The data for the upcoming election wasn't up yet, so I chose 2014.) I exported the CSV, parsed the lat/lon into separate columns, imported into QGIS as a Delimited Text Layer, and ran PIP to generate "Polls in Polygons":



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

2014 Los Angeles County polling places by Neighborhood

Polls in Polygons (https://tangrams.github.io/tangram-frame/?noscroll&url=https://mapzen-assets.s3.amazonaws.com/resources/coloring-choropleths/la_polls_choropleth.yaml#13/34.0431/-118.3712)

The source for this map **is also available in Tangram Play** (<https://mapzen.com/tangram/play/?scene=https%3A%2F%2Fmapzen-uploads.s3.amazonaws.com%2Fplay%2F262792e74b86d8cb772a3f0fd83c4328%2Fla-polls%2Fscene.yaml#14.0792/34.0155/-118.2812>).

In this map, I decided to color the polygons using `OrRd`, a **ColorBrewer** (colorbrewer2.org) palette.

```
neighborhoods:
  data: { source: polygons_in_polls}
  # filter: function() { return feature.poll_count > 0; }
  draw:
    lines:
      order: 500
      color: [0,0,0,0.7]
      width: [[7,0.1px],[18,3px]]
    choropleth:
      order: 500
      color: |
        function() {
          var count = feature.poll_count;
          var OrRd = ['fef0d9', 'fdcc8a', 'fc8d59', 'e34a33', 'b30000'];
          var color =
            count = 0 ? '000' :
            count < 15 ? OrRd[0] :
            count < 30 ? OrRd[1] :
            count < 60 ? OrRd[2] :
            count < 120 ? OrRd[3] :
            OrRd[4] ;
          return "#" + color;
        }
      
```

Note that Tangram automatically handles and manages collisions between dots, drawing a representative number at each zoom level – this gives an accurate representation of the overall distribution of the polling places. It will also move labels around their points to fit as much text as possible. And as with the pool map, more details about the polling places appears as you zoom in.

The 4000 polling places are fairly evenly distributed, as one would hope. Other cities in L.A. County, like Long Beach, Santa Monica and Pasadena, had more polling places and were redder than L.A. neighborhoods – while some of this is simply due to size, importing population by neighborhood and calculating and displaying people per poll per boundary is future choroplething opportunity.

While fairly basic, we hope these demos will prove useful if you want to build a choropleth map using Tangram. If you have any questions or suggestions, email us at **hello@mapzen.com** (<mailto:hello@mapzen.com>) or drop us a line **on Twitter** (<https://twitter.com/intent/tweet?@mapzen+choropleths+are+awesome>)!

· 23 September 2016 ·



John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

Announcing version 1.1 of the Mapzen Android SDK

mobile (/tag/mobile) **android** (/tag/android)

The Mapzen mobile team is here to bring open source to the mobile ecosystem. We build open source alternatives to the closed source native mapping libraries and make them easy to hook into Mapzen's open data.

Today we are announcing **version 1.1 of the Mapzen Android SDK** (<https://github.com/mapzen/android/releases/tag/v1.1.0>). This is the first major update to our Android SDK since we **launched it last April** (<https://mapzen.com/blog/android-sdk/>).

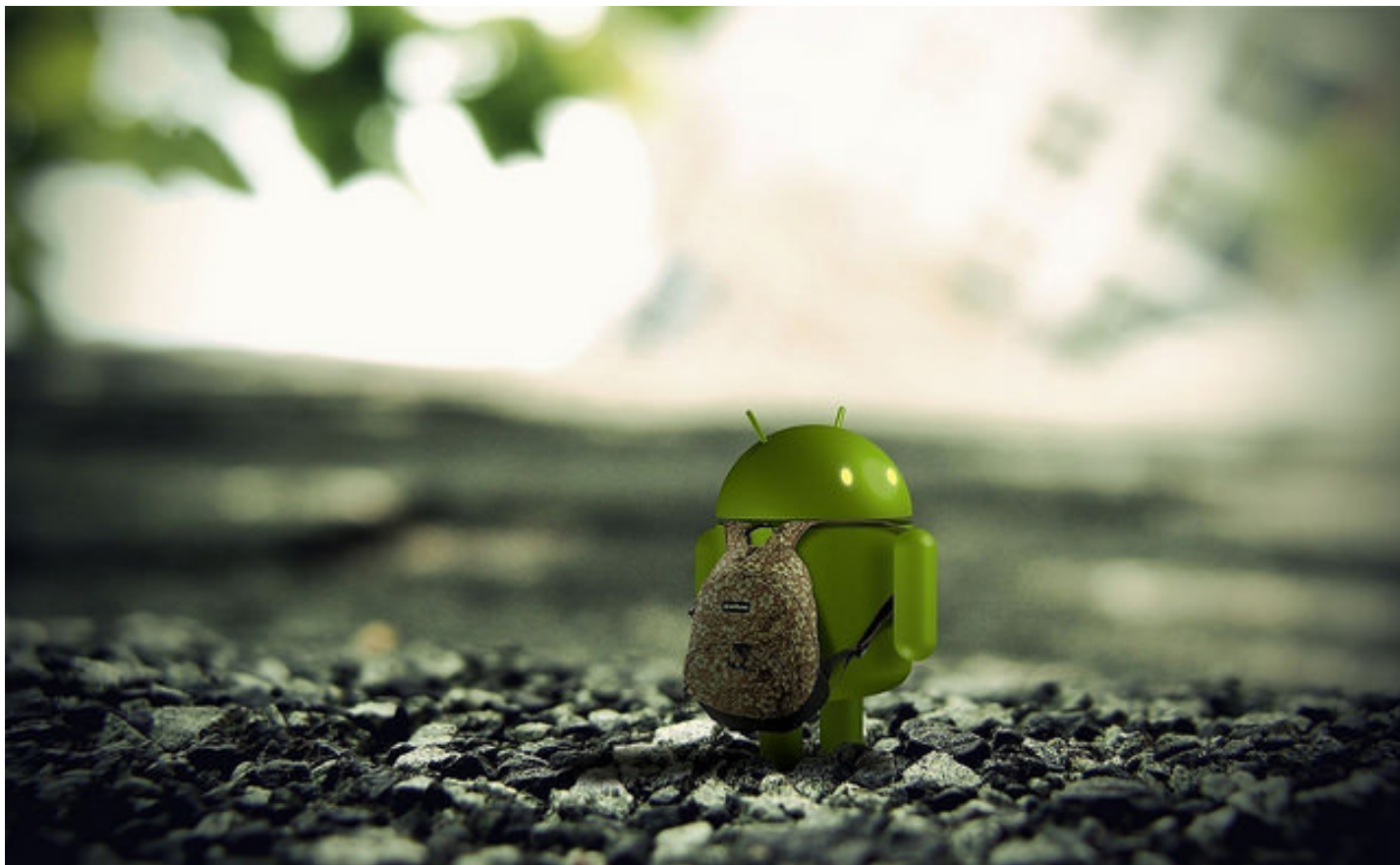


image via 00はがはがはが, CC BY 2.0 (<https://flic.kr/p/o8PVDu>)

Search (**Pelias Android SDK** (<https://github.com/pelias/pelias-android-sdk>)) and routing (**On the Road** (https://github.com/mapzen/on-the-road_android)) are now fully integrated into the SDK and the mapping experience. We also rolled out major updates to map rendering (**Tangram ES** (<https://github.com/tangrams/tangram-es>)) and location services (**Lost** (<https://github.com/mapzen/lost>)).

Leading up to **Droidcon NYC** (<http://droidcon.nyc/>), we'll be posting everyday this week about a different component of the Mapzen Android SDK and how it can be helpful to your location based apps. We hope you'll follow along and then come say hi at Droidcon.

- **Monday (9/26) Announcing version 1.1 of the Mapzen Android SDK** (<https://mapzen.com/blog/android-sdk-whats-new/>)
- **Tuesday (9/27) Making Better Maps with Tangram on Android** (<https://mapzen.com/blog/better-maps-with-tangram/>)
- **Wednesday (9/28) Mapzen Search for Android** (<https://mapzen.com/blog/mapzen-search-for-android/>)
- **Thursday (9/29) Routing and navigation with Mapzen Turn-by-Turn** (<https://mapzen.com/blog/android-routing-and-navigation>)
- **Friday (9/30) Open source location services with Lost 2.0** (<https://mapzen.com/blog/open-source-location-services-with-lost>)

· 26 September 2016 ·



Mike Cunningham

I'm Street View famous in two cities and I do product @mapzen.



Chuck Greb

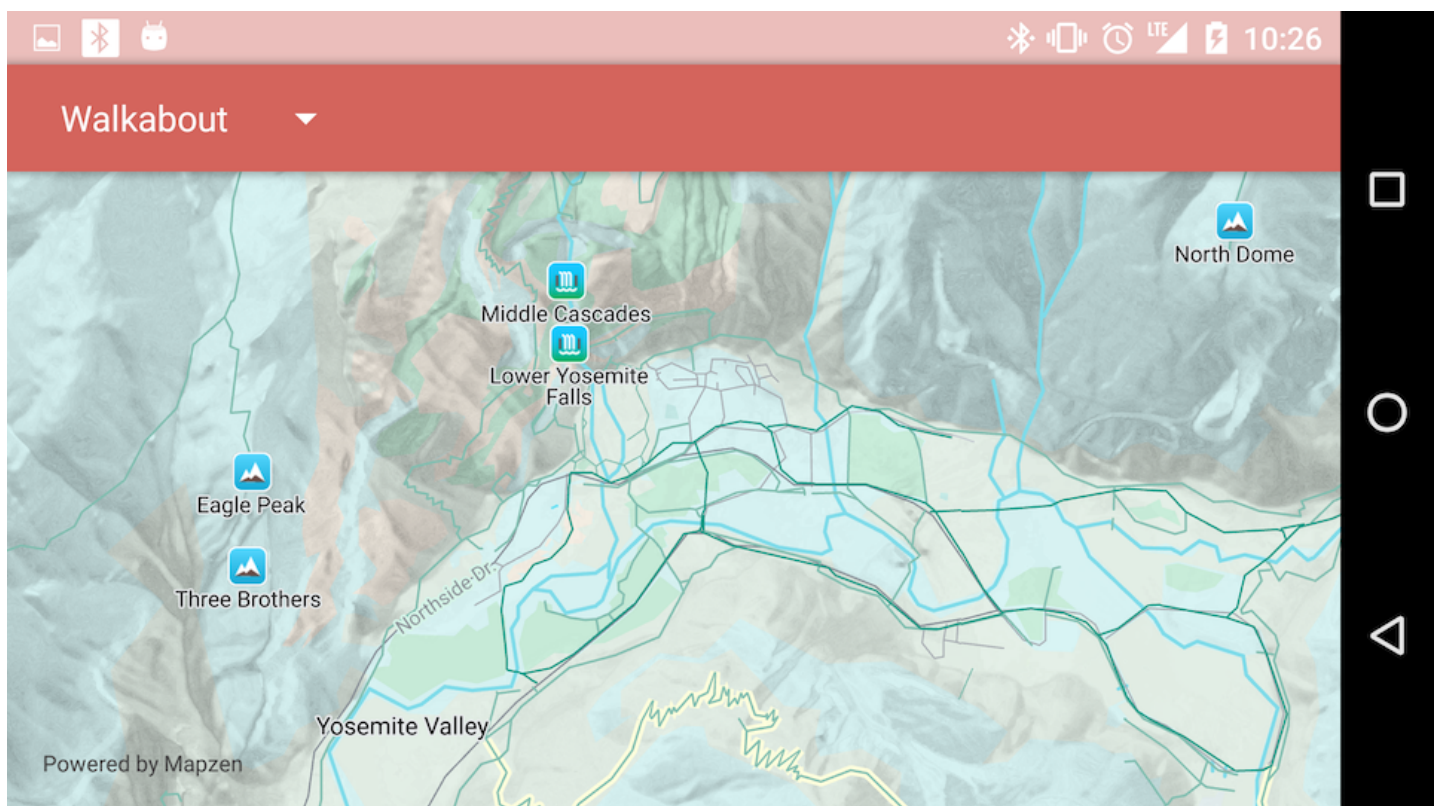
Chuck is an open source enthusiast, test-driven evangelist, and clean code connoisseur of all things mobile.

Making Better Maps with Tangram on Android

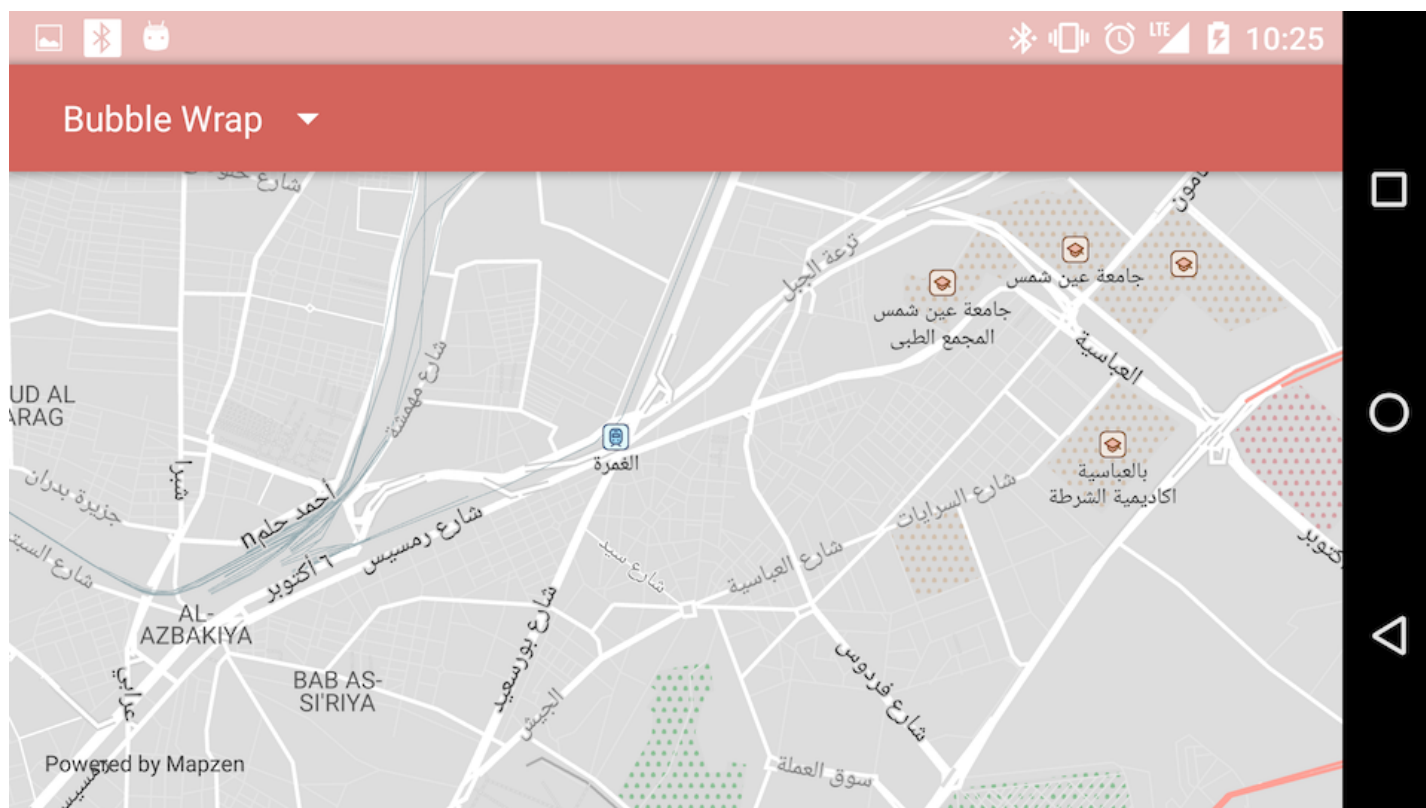
mobile (/tag/mobile) **android** (/tag/android) **tangram** (/tag/tangram)

One of the four symbiotic hearts of the Mapzen Android SDK is the rendering engine that paints maps, markers, and routes onto a screen. The rendering engine has the job of visually communicating the deluge of data that a map contains about any part of the world, and doing it quickly and efficiently so that it's delightful to use. This is a hard problem, but one that's essential to a good mapping experience.

In the Mapzen Android SDK we chose to use our home-grown cross-platform map renderer, **tangram-es** (<https://github.com/tangrams/tangram-es>). There are three main reasons that tangram-es was the best fit for our needs (and maybe yours too). First, it's entirely open-source, which is important to the Mapzen commitment to an open mapping ecosystem. Second, it is relatively agnostic to the format and contents of the map data you use, so you're not locked into Mapzen's vector tiles. Third, it has a *ludicrously* flexible **styling system** (<https://mapzen.com/documentation/tangram/Scene-file/>) that lets us make some very handsome and effective cartography.



With tangram-es we have a unique set of cartographic tools, including a text layout and rendering engine that we've built from the ground up to produce high-quality text **handling the many different features of many scripts** (<http://rishida.net/scripts/featurelist/>), even those with text-shaping (like Hindi), right-to-left layout (like Hebrew), or both (like Arabic).



Flexibility is great, but it's like a double-edged sword — or at least like a really sharp cheese grater that makes you nervous when you use it. Sometimes you don't need precise control over every aspect of a map: you just want it to work and look good. The Mapzen SDK has you covered so you can focus on the cool stuff that you're doing in the rest of your app. Our expert cartography team has put together a global set of vector tiles and a comprehensive map style that are both available automatically when you use the SDK. We also provide handy bits like an automatic current location marker (this uses LOST, our location provider, which you'll hear more about soon!).

Getting a map view into your app with the Mapzen Android SDK is straightforward and should be familiar if you've used other mapping SDKs. We've documented it all here:

<https://mapzen.com/documentation/android/getting-started/>
(<https://mapzen.com/documentation/android/getting-started/>)

Maybe you're the type of developer who likes really sharp swords and cheese graters. You might prefer to have full control over all the details of the rendering engine. Fortunately, the full array of tangram-es features is at your fingertips when you use the Mapzen Android SDK. Using

`MapzenMap$getMapController()` you can access the tangram-es instance that powers the graphics in the SDK.

If you're interested in using tangram-es on Android by itself, it's easily accessible through Gradle as a Maven dependency. In your `build.gradle` file, just add `compile "com.mapzen.tangram:tangram:$tangram_version"` to the `dependencies` section of your project with the latest tangram-es version (**check Maven Central** (<http://search.maven.org/#search%7Cga%7C1%7Ctangram>)).

There's a lot more to say about map rendering, but this is just *one* of the tools you get when you use the Mapzen Android SDK. Stay tuned for the rest of this week to hear about our search, routing, and location tools!

- **Monday (9/26) Announcing version 1.1 of the Mapzen Android SDK**
(<https://mapzen.com/blog/android-sdk-whats-new/>)
- **Tuesday (9/27) Making Better Maps with Tangram on Android**
(<https://mapzen.com/blog/better-maps-with-tangram/>)
- **Wednesday (9/28) Mapzen Search for Android** (<https://mapzen.com/blog/mapzen-search-for-android/>)
- **Thursday (9/29) Routing and navigation with Mapzen Turn-by-Turn**
(<https://mapzen.com/blog/android-routing-and-navigation>)
- **Friday (9/30) Open source location services with Lost 2.0**
(<https://mapzen.com/blog/open-source-location-services-with-lost>)

· 27 September 2016 ·



Varun Talwar

Graphics Engineer. C++, Graphics and Physics Simulation enthusiast. Worked on Animation Movies.

Mapzen Search for Android

mobile (/tag/mobile) **android** (/tag/android) **search** (/tag/search)

The **Mapzen Android SDK** (<https://mapzen.com/documentation/android/>) provides developers with an easy way to integrate **Mapzen Search** (<https://mapzen.com/documentation/search/>) into their Android applications.

Mapzen Search is a modern, geographic search service based entirely on open-source tools and powered entirely by open data. You might use this functionality in any app that has a geographic element, including ones that deliver goods, locate hotels or venues, or even provide local weather forecasts.



image via Charles Clegg, CC BY-SA 2.0 (<https://flic.kr/p/jie2n2>)

Mapzen Search is powered by the **Pelias open source geocoder** (<http://pelias.io/>) project. Similarly, the Mapzen Android SDK makes use of the open source **Pelias Android SDK** (<https://github.com/pelias/pelias-android-sdk>) library. The Pelias Android SDK provides a convenient interface to the Mapzen Search API and is built using popular networking components like **OkHttp** (<http://square.github.io/okhttp/>) and **Retrofit** (<http://square.github.io/retrofit/>).

The Android SDK supports full text search, autocomplete, and reverse geocoding. Or you can use `PeliasSearchView`, a drop-in extension to the Android `SearchView` component that supports search and autocomplete out of the box. There is also a convenient interface to show search results on a Mapzen map.

Don't forget to **grab a free Mapzen API key** (<https://mapzen.com/developers/>) before you get started. Also you can check out the docs for the **Mapzen Android SDK** (<https://mapzen.com/documentation/android/>) and the **SDK demo app** (<https://github.com/mapzen/android/tree/master/sample>).

Make sure to check back tomorrow when we talk about mobile routing and navigation with Mapzen Turn-by-Turn.

- **Monday (9/26) Announcing version 1.1 of the Mapzen Android SDK** (<https://mapzen.com/blog/android-sdk-whats-new/>)
- **Tuesday (9/27) Making Better Maps with Tangram on Android** (<https://mapzen.com/blog/better-maps-with-tangram/>)
- **Wednesday (9/28) Mapzen Search for Android** (<https://mapzen.com/blog/mapzen-search-for-android/>)
- **Thursday (9/29) Routing and navigation with Mapzen Turn-by-Turn** (<https://mapzen.com/blog/android-routing-and-navigation>)
- **Friday (9/30) Open source location services with Lost 2.0** (<https://mapzen.com/blog/open-source-location-services-with-lost>)

· 28 September 2016 ·



Chuck Greb

Chuck is an open source enthusiast, test-driven evangelist, and clean code connoisseur of all things mobile.

© 2017 Mapzen

Routing and Navigation with Mapzen Turn-by-Turn

mobile (/tag/mobile) **android** (/tag/android) **routing** (/tag/routing)

One of the **four pillars** (<https://mapzen.com/blog/android-sdk-whats-new>) of the **Mapzen Android SDK** (<https://github.com/mapzen/android>) is **Mapzen Turn-by-Turn** (<https://mapzen.com/products/turn-by-turn>), our hosted routing service. Mapzen Turn-by-Turn is powered by **Valhalla** (<https://github.com/valhalla>), our routing engine which is built on open data from OpenStreetMap and Transitland. Mapzen Turn-by-Turn is incredibly flexible and allows for on-the-fly customization of **costing models** (<https://mapzen.com/documentation/mobility/turn-by-turn/api-reference/#costing-models>) and **options** (<https://mapzen.com/documentation/mobility/turn-by-turn/api-reference/#costing-options>).



image via ScotEaster2013_188, CC BY-SA 2.0 (<https://flic.kr/p/eazM8Q>)

In the Mapzen Android SDK, the interface to Mapzen Turn-by-Turn is **On-The-Road** (https://github.com/mapzen/on-the-road_android), our Valhalla wrapper. On-The-Road handles client-side challenges of routing such as snapping the device location to the closest nearby road, distance calculations, and turn prompting. With On-The-Road, developers can request a route for a set of locations. This object provides information about the user's progress such as when the user starts moving along the route. It also is aware of when the user is lost and the route needs to be recalculated. In addition, the route knows when instructions are approached and should be called out to the developer or when the user has navigated to the final location.

On-The-Road depends on a small set of external dependencies (Kotlin, OkHttp, Retrofit) and can be used with other tools in the Mapzen suite or as an independent library. In **EraserMap** (<https://mapzen.com/blog/erasermap-beta/>), our **open source** (<https://github.com/mapzen/eraser-map>), privacy focused navigation application, we use On-The-Road along side of Tangram, Pelias and Speakerbox. We've heard about **Tangram** (<https://mapzen.com/blog/2016-9-27-better-maps-with-tangram>) and **Pelias** (<https://mapzen.com/blog/2016-9-28-mapzen-search-for-android>) earlier this week, but what is **Speakerbox** (<https://github.com/mapzen/speakerbox>)? Speakerbox is our wrapper around Android's TextToSpeech class. It provides a simple interface for playing and remixing text and depends only on the Android support library.

Recently, we've made some exciting improvements to both On-The-Road and Speakerbox. We migrated On-The-Road away from **The Open Source Routing Machine** (<http://project-osrm.org/>) to Mapzen Turn-by-Turn, added more information around instructions and routing events, and added support for a new costing model. In Speakerbox, we added support for **audio ducking** (<https://en.wikipedia.org/wiki/Ducking>) and the ability to execute code when Speakerbox begins playing text, finishes playing it, and when an error, such as the voice data hasn't finished downloading, occurs while trying to play it.

We are excited about the improvements to these two libraries and hope that they make developing routing applications easier and more intuitive. Head over to the **Mapzen Android SDK** (<https://github.com/mapzen/android>) to start using them today. As always, if you have suggestions on how to further improve these libraries we would love to **hear from you** (<https://github.com/mapzen/android/issues/new>).

- **Monday (9/26) Announcing version 1.1 of the Mapzen Android SDK** (<https://mapzen.com/blog/android-sdk-whats-new/>)
- **Tuesday (9/27) Making Better Maps with Tangram on Android** (<https://mapzen.com/blog/better-maps-with-tangram/>)

- **Wednesday (9/28) Mapzen Search for Android (<https://mapzen.com/blog/mapzen-search-for-android/>)**
- **Thursday (9/29) Routing and navigation with Mapzen Turn-by-Turn (<https://mapzen.com/blog/android-routing-and-navigation>)**
- **Friday (9/30) Open source location services with Lost 2.0 (<https://mapzen.com/blog/open-source-location-services-with-lost>)**

· 29 September 2016 ·



Sarah Lensing

mobile developer. startup junkie. athlete. older sister. nyu alum. vermonter. maker of beef jerky and other random food items.

© 2017 Mapzen

Open source location services on Android with Lost 2.0

mobile (/tag/mobile) **android** (/tag/android)

Lost is a drop-in replacement for Google Play services **Location APIs** (<http://developer.android.com/google/play-services/location.html>) which offers open source alternatives to the **FusedLocationProviderApi** (<https://developer.android.com/reference/com/google/android/gms/location/FusedLocationProviderApi.html>), **GeofencingApi** (<https://developers.google.com/android/reference/com/google/android/gms/location/GeofencingApi.html>), and **SettingsApi** (<https://developers.google.com/android/reference/com/google/android/gms/location/SettingsApi.html>). It depends only on the Android SDK and makes calls directly to the **LocationManager** (<https://developer.android.com/reference/android/location/LocationManager.html>).

Over the last few months we've made significant updates to Lost, greatly expanding its functionality and bringing it closer to API parity with its closed-source counterpart. (For more background and history about the Lost project you can check out the **original blog post** (<https://mapzen.com/blog/lets-get-lost/>) when we first launched the library two years ago.)



crop of image via Natasha Habeduš, CC BY-SA 2.0 (<https://flic.kr/p/fsFHEc>)

Location services in the Mapzen Android SDK

Lost powers all location services in the **Mapzen Android SDK**

(<https://mapzen.com/documentation/android/>) and is responsible for things like showing your current location on the map, influencing search results, and tracking your progress as you navigate along a route.

Background location updates via `PendingIntent`

Lost now supports requesting background location updates. Developers can use the `FusedLocationProviderApi` to request updates via `PendingIntent`. We recommend having the `PendingIntent` launch an `IntentService` to handle work that needs to be done in response to location changes.

Geofencing API

The new 2.0 release also includes basic support for geofencing, allowing developers to receive notifications when the device's current location enters or exits a specified circular region. The `GeofencingApi` is an important new feature of Lost and opens many new possibilities: developers of location-based apps can now show special content in particular regions, and games can change levels upon entrance of an area.

Settings API

Users often change their location settings to conserve battery or increase privacy. When this occurs, an application's location requirements may no longer be met. With an implementation of the `SettingsApi` Lost now includes a mechanism to check the device's current location and bluetooth settings, and optionally update them if they do not meet the requirements of a client application. All of these new features come with almost no overhead, making integration quite seamless.

Location Availability

Sometimes you don't want to receive ongoing location updates but instead want to know, at discrete points in time, if a location is available. Using `LocationAvailability` you can query to find out if any location providers are enabled, and if there is a last known location to report.

Permissions

We want all of these new features to have as little impact on developers as possible, which is why we have removed permissions from the library manifest. All methods which require location or bluetooth permissions are now annotated, giving developers the freedom to choose which APIs and permissions they would like to include in their applications.

Documentation and sample app

If you are currently using Lost 1.X we put together a small **migration guide** (<https://github.com/mapzen/lost/blob/master/docs/upgrade-1x-2.md>) to help you upgrade your app to use Lost 2.0. For more information check out the **documentation** (<https://github.com/mapzen/lost/tree/master/docs>) and **sample app** (<https://github.com/mapzen/lost/tree/master/lost-sample>). **Let us know** (<mailto:hello@mapzen.com?subject=lost>) what you build with Lost and all the other new features in the Mapzen SDK!

- **Monday (9/26) Announcing version 1.1 of the Mapzen Android SDK** (<https://mapzen.com/blog/android-sdk-whats-new/>)
- **Tuesday (9/27) Making Better Maps with Tangram on Android** (<https://mapzen.com/blog/better-maps-with-tangram/>)
- **Wednesday (9/28) Mapzen Search for Android** (<https://mapzen.com/blog/mapzen-search-for-android/>)
- **Thursday (9/29) Routing and navigation with Mapzen Turn-by-Turn** (<https://mapzen.com/blog/android-routing-and-navigation>)
- **Friday (9/30) Open source location services with Lost 2.0** (<https://mapzen.com/blog/open-source-location-services-with-lost>)

· 30 September 2016 ·



Chuck Greb

Chuck is an open source enthusiast, test-driven evangelist, and clean code connoisseur of all things mobile.



Sarah Lensing

mobile developer. startup junkie. athlete. older sister. nyu alum. vermonter. maker of beef jerky and other random food items.

© 2017 Mapzen

The Mapzen Libpostal API

data (/tag/data)

We are happy to announce that access to the **Libpostal** (<https://github.com/openvenues/libpostal>) address parsing and expansion services are now available via the **Mapzen API** (<https://mapzen.com/developers>).

Al Barrentine (<https://twitter.com/albarrentine>) (Libpostal's author) has written an **exhaustive blog post** (<https://mapzen.com/blog/inside-libpostal/>) describing what Libpostal is and how it works. The post is aimed at a technical audience so we'll just excerpt the short version here:

Libpostal uses machine learning and is informed by tens of millions of real-world addresses from OpenStreetMap. The entire pipeline for training the models is open source. Since OSM is a dynamic data set with thousands of contributors and the models are retrained periodically, improving them can be as easy as contributing addresses to OSM.

Each country's addressing system has its own set of conventions and peculiarities and libpostal is designed to deal with practically all of them. It currently supports normalizations in 60 languages and can parse addresses in more than 100 countries. Geocoding using libpostal as a preprocessing step becomes drastically simpler and more consistent internationally.

Who's On First (<https://whosonfirst.mapzen.com/>) is already using Libpostal for its internal editorial tool (more about that from Dan **over here** (<https://mapzen.com/blog/boundary-issues-properties/>)) and the **Search** (<https://mapzen.com/products/search/>) team has started work on **integrating Libpostal with the Pelias geocoder** (http://pelias.io/milestones/libpostal_integration/).

Address

EXTRACT PROPERTIES

Options

THIS IS CLOSED

THIS IS DEPRECATED

Now you can use Libpostal in your projects too, simply by calling the **Mapzen API** (<https://mapzen.com/developers/>)!

Some of you may have noticed that the newly parsed address string has grown a postal code, in the example above. This is not something that Libpostal does but rather the result of another little piece of data-magic we'll talk about more soon.

cURL or it didn't happen

To unwind and normalize **all the possible variations for parts of an address string** (<https://libpostal.mapzen.com/expand?address=475+Sansome+St+San+Francisco+CA>) that may be encoded using abbreviations, or some other context-specific short-hand, you would call the `/expand` endpoint. Like this:

```
curl -s 'https://libpostal.mapzen.com/expand?address=475+Sansome+St+San+Francisco+CA' | p
[
  "475 sansome saint san francisco california",
  "475 sansome saint san francisco ca",
  "475 sansome street san francisco california",
  "475 sansome street san francisco ca"
]
```

To **explode an address string in to each of its component parts**

(<https://libpostal.mapzen.com/parse?address=475+Sansome+St+San+Francisco+CA>) you would call the `/parse` endpoint. Like this:

```
curl -s 'https://libpostal.mapzen.com/parse?address=475+Sansome+St+San+Francisco+CA' | py
[
  {
    "label": "house_number",
    "value": "475"
  },
  {
    "label": "road",
    "value": "sansome st"
  },
  {
    "label": "city",
    "value": "san francisco"
  },
  {
    "label": "state",
    "value": "ca"
  }
]
```

By default both **Libpostal** (<https://github.com/openvenues/libpostal>) and the Libpostal API return results a list of dictionaries, each containing a `label` and `value` key. This is because there are occasions when a given key may have multiple values, for example an address that contains a cross-street.

If you would prefer **to have API results returned as a simple dictionary**

([https://libpostal.mapzen.com/parse?](https://libpostal.mapzen.com/parse?address=475+Sansome+St+San+Francisco+CA&format=keys)

address=475+Sansome+St+San+Francisco+CA&format=keys) with labels as keys and values as lists of possible strings simply append the `format=keys` parameter. Like this:

```
curl -s 'https://libpostal.mapzen.com/parse?address=475+Sansome+St+San+Francisco+CA&format=json'
{
  "city": [
    "san francisco"
  ],
  "house_number": [
    "475"
  ],
  "road": [
    "sansome st"
  ],
  "state": [
    "ca"
  ]
}
```

Tell me more

The Mapzen Libpostal API is available for testing and experimental use without the need for an API key so you can get started testing addresses right away. As with all keyless API access **usage is limited** (<https://mapzen.com/documentation/overview/>) so if you want to do more serious work we encourage you to sign-up for a Mapzen API key. It only takes a minute (or two) and you can **get yours here** (<https://mapzen.com/developers>).

Complete documentation for the Mapzen Libpostal API is available from:

<https://github.com/whosonfirst/go-whosonfirst-libpostal/blob/master/docs/index.md>
(<https://github.com/whosonfirst/go-whosonfirst-libpostal/blob/master/docs/index.md>)

*The **splash image** (https://mapzen-assets.s3.amazonaws.com/images/libpostal_api/libpostal_escape_map.png) for this blog post is a crop from a **World War II Escape map** (<https://collection.cooperhewitt.org/objects/18639109/>), courtesy the Cooper Hewitt Smithsonian Design Museum.*

· 04 October 2016 ·



Aaron Straup Cope

Aaron is happiest in the presence of olive oil.

© 2017 Mapzen

Boundary Issues: Editing Properties in Who's On First Records

whosonfirst (/tag/whosonfirst) **boundaryissues** (/tag/boundaryissues) **data** (/tag/data)

Who's On First records (<https://whosonfirst.mapzen.com/>), encoded as GeoJSON flat files, have always supported a failure scenario where you can open them up in Microsoft Word, or TextEdit, or *your favorite text editor here* and save the changes. Or, if you're feeling *even lazier*, you could **go to GitHub** (<https://github.com/whosonfirst-data/whosonfirst-data>) and click on the little pencil icon to modify a record in your browser.

[whosonfirst-data-venue-us-ca](#) / [data](#) / [100](#) / [818](#) / [405](#) / [1](#) / 1008184051.geojson  or [cancel](#)

<> Edit file
Preview changes
Spaces
2
No wrap

```

1 {
2   "id": 1008184051,
3   "type": "Feature",
4   "properties": {
5     "edtf:cessation": "2016-09-18",
6     "edtf:inception": "2016-08",
7     "geom:area": 0,
8     "geom:bbox": "-122.411529,37.743715,-122.411529,37.743715",
9     "geom:latitude": 37.743715,
10    "geom:longitude": -122.411529,
11    "iso:country": "US",
12    "mz:categories": [
13      "do_and_see:attraction=public_art"
14    ],
15    "mz:hours": [],
16    "mz:is_current": 0,
17    "name:eng_x_preferred": [
18      "Poop Emoji Rock"
19    ],
20    "name:eng_x_variant": [

```

As it turns out, these methods can be cumbersome, so we've been developing an internal-facing (for now) web-based editor for Who's On First records called **Boundary Issues** (<https://github.com/whosonfirst/whosonfirst-www-boundaryissues/>) (shout out to former Mapzen-er **Ingrid Burrington** (<https://twitter.com/lifewinning>) for the name—also, hey, go **buy her book** (<http://www.mhpbooks.com/books/networks-of-new-york/>)). And to be clear, yes, this is a public blog post about a not-yet-public editing tool. It is still just a little too early to open access up to everyone, but we will get there.

Boundary Issues does not yet edit polygon boundaries in the way that **OpenStreetMap's iD editor** (<http://www.openstreetmap.org/edit?editor=id>) does. It will do that someday, but right now it's primarily for editing GeoJSON *properties*. Here is an excerpt of the **GeoJSON document** (<https://whosonfirst.mapzen.com/data/100/818/405/1/1008184051.geojson>) for the **Poop Emoji Rock** (<https://bernalwood.com/2016/08/29/bernal-rock-transformed-into-beloved-poop-emoji/>):

```
{
  "type": "Feature",
  "geometry": {
    "coordinates": [
      -122.411529,
      37.743715
    ],
    "type": "Point"
  },
  "properties": {
    "geom:latitude": 37.743715,
    "geom:longitude": -122.411529,
    "wof:id": 1008184051,
    "wof:name": "Poop Emoji Rock",
    "wof:parent_id": 420780697,
    "wof:placetype": "venue"
  }
}
```

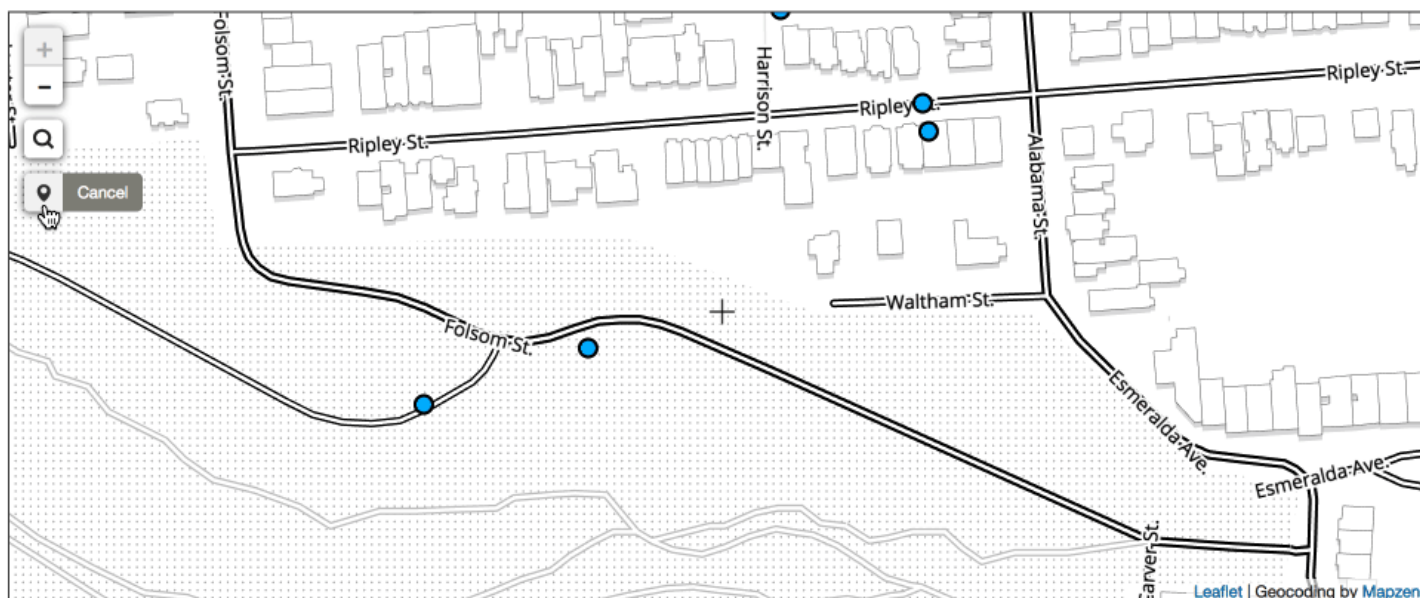
If you're unfamiliar with the Poop Emoji Rock, it is a frequently painted rock that was recently made to look like Unicode character U+1F4A9 "PILE OF POO" (also known as 🐞). As you might expect, it **has a Who's On First record** (<https://twitter.com/alloftheplaces/status/770793279969374208>).



Image courtesy of **Telstar Logistics** (<https://bernalwood.com/2016/08/29/bernal-rock-transformed-into-beloved-poop-emoji/>).

GeoJSON is a specific kind of JSON (a file format for structuring data), which **specifies** (<https://tools.ietf.org/html/rfc7946>) that we can expect a handful of predictable top-level properties. GeoJSON makes it easy for a computer to keep descriptions about *where a place is* (the `geometry`) separate from descriptions about the *qualities of the place* (the `properties`).

The very first feature of Boundary Issues was for choosing a place's geometry: *click on the map to set the position*. (Thanks to **Leaflet.draw** (<https://github.com/Leaflet/Leaflet.draw>) for the add-a-pin UI.)



map is centered at 37.743715, -122.411529 #18

As for what goes in the `GeoJSON properties` list, it is very open-ended. **Who's On First** (<https://mapzen.com/blog/who-s-on-first/>) provides some conventions about **what and how to encode things in the GeoJSON properties** (<https://github.com/whosonfirst/whosonfirst-properties>). This is meant to maximize the compatibility of GeoJSON records between a broad variety of applications.

For example, here is how you might add tags to a record using Boundary Issues.

wof:name	Poop Emoji Rock
wof:parent_id	420780697
wof:placetype	venue
wof:superseded_by	Add an item +
wof:supersedes	Add an item +
wof:tags	emoji -
	Add an item +

Notice that we're using property *namespaces*. Instead of specifying a `tags` property, we use `wof:tags`, meaning it's a part of the **Who's On First namespace** (<https://github.com/whosonfirst/whosonfirst->

properties/blob/master/properties/wof.md). This lets us keep things that are common to *all WOF records* like `wof:name` separate from *Mapzen opinionated things* like `mz:categories`. This namespacing is in the same **spirit** (<https://en.wikipedia.org/wiki/Triplestore>) as the **Semantic Web** (https://en.wikipedia.org/wiki/Semantic_Web), but without the verbosity of **RDF** (https://en.wikipedia.org/wiki/Resource_Description_Framework#Examples).

The namespaces also offer a natural way to group related properties in the user interface.

edtf click to collapse

edtf:cessation	2016-09-18
edtf:inception	2016-08

geom click to expand

iso click to expand

mz click to expand

name click to expand

Unlike the more freestyle approach of **geojson.io** (<http://geojson.io/>), Boundary Issues has built-in scaffolding that guides how properties get encoded. The user interface is saying “here are some of the things you can type in.” For example, what are the **names** (<https://github.com/whosonfirst/whosonfirst-names>) people are known to call this place? Are there **concordances to records in other databases** (<https://github.com/whosonfirst/whosonfirst-properties/blob/master/properties/wof.md#concordances>) we can hold hands with?

Boundary Issues also keeps a short list of *minimal viable properties* that are required to make a Who's On First record. When creating a **venue placetype** (<https://github.com/whosonfirst/whosonfirst-placetypes/blob/master/placetypes/venue.json>) you can pretty much drop a pin on the map, type in a `wof:name` value, and you've got yourself a WOF record. All the other properties are optional, and will be assigned reasonable default values.

We turned to **JSON Schema** (<http://json-schema.org/>), *yet another flavor of JSON*, to define how different properties get encoded. JSON Schema was originally created as a way of declaring guidelines that say *this is VALID* or *this is INVALID*. Using it for that purpose does have **some complications** (<https://www.tbray.org/ongoing/When/201x/2016/04/30/JSON-Schema-funnies>), but the specification files themselves can be read by a variety of programming languages. It lets us build a basic scaffolding for records that isn't tied into any one particular application.

We can use the schema to construct a brand new empty record, and to nudge each of its properties into their intended data types (strings, numbers, lists, etc.), and provide a default value when we need one. Perhaps we will use our JSON Schema for validating records some day.

Here's an excerpt from **our schema** (<https://github.com/whosonfirst/whosonfirst-json-schema/blob/master/schema/whosonfirst.schema>) (derived from **this sample GeoJSON schema** (<https://github.com/fge/sample-json-schemas/blob/master/geojson/geojson.json>)):

```
{
  "properties": {
    "wof:id": {
      "type": "integer"
    },
    "wof:parent_id": {
      "type": "integer",
      "default": -1
    },
    "wof:name": {
      "type": "string"
    }
  }
}
```

The way the property-editing interface *behaves* requires an additional set of **guidelines** (https://github.com/whosonfirst/whosonfirst-www-boundaryissues/blob/master/www/include/lib_wof_render.php#L44-L133). These rules *are* baked into the source code (since they're specific to Boundary Issues), and they let us specify things on a per-property basis like:

- Is the property *editable*? If not, the property should be set by software automatically.
- Is the property *deletable*? Which is really just another way of saying “this property is required, *it must have a value of some sort.*”

- Is the property *visible*? Almost all properties are visible in the list, but sometimes it's helpful to hide them from the editing interface.
- Is the property on the short list of *minimum viable properties*?

Collectively these data types, default values, and editing behaviors get baked into an HTML `<form>` element, with various clues for the JavaScript code about how it should treat each property.

```
<div class="json-schema-field json-schema-required">
  <input type="text" name="properties.wof:parent_id" value="-1"
    disabled="disabled" data-type="integer">
</div>
```

Minimum viable properties click to collapse

wof:id	1008184051
wof:name	Poop Emoji Rock
wof:parent_id	420780697
wof:placetype	venue

Add a new property

In the case of `wof:parent_id`, you might wonder how it can be both *required* and also *disabled*? With these constraints, how could the value ever change from the default value? The answer is in our growing assortment of bespoke editing interfaces, discussed below.

In the case of `wof:parent_id` and `wof:hierarchy`, these values are chosen automatically thanks to our internal **point in polygon service** (<https://github.com/whosonfirst/go-whosonfirst-pip>). After dropping a pin on the map, the code consults with the point in polygon service, then suggests a potential hierarchy (or *hierarchies*, **there can be more than one** (<https://github.com/whosonfirst/whosonfirst-placetypes#hierarchies>)).

Hierarchy

Parent: **Sutrito Canine Republic**

420780697

- continent: **North America** **102191575**
- country: **United States** **85633793**
- county: **San Francisco** **102087579**
- locality: **San Francisco** **85922583**
- microhood: **Sutrito Canine Republic** **420780697**
- neighbourhood: **Bernal Heights** **85865945**
- region: **California** **85688637**

The hierarchy interface doesn't actually offer any new user-facing controls, instead it just kind of **does stuff** (<http://lifewinning.com/tag/magic/>), encoding properties behind the scenes based on the chosen latitude and longitude coordinates. There are some additional bespoke property interfaces within the UI that make it easier to express things that don't sit neatly inside of text input boxes. Recent additions include a multi-language name editor, an address parsing widget, a way of encoding a venue's open and closing hours, and a way of encoding venue categories. We hope some of these UI widgets—such as the address parsing one—will eventually get extracted out into a generalized JavaScript library for use outside of Boundary Issues.

Address

Enter full address here



EXTRACT PROPERTIES

Options

THIS IS CLOSED

THIS IS DEREGATED

Who's On First Life Cycle Documentation

`whosonfirst` `(/tag/whosonfirst)` `data` `(/tag/data)`



Who's On First (<https://whosonfirst.mapzen.com>) is Mapzen's gazatteer of places. Within Who's On First, each place is represented as a record with a unique and stable identifier that we call the **wof:id**. Because our gazetteer tracks changes to places over time we create new records when places experience significant events (and point the old record forward to the new record's **wof:id**).

The Who's On First **wof:id** comes with various levels of complexity and convention; the guidelines around the **wof:id** can be found in our new **Who's On First ID Life Cycle Documentation** (https://github.com/whosonfirst/whosonfirst-cookbook/blob/master/definition/wof:id_lifecycle.md).

So, why is documenting these rules is so important?

It is important to document the rules around **wof:id** because sometimes a place experiences a change so significant that it warrants “snapshotting” it in the past preserving its original **wof:id** so that it (the place) can move forward with a new **wof:id**.

For example: St. Petersburg, Russia became Leningrad for a while, then changed back to St. Petersburg. These are all the same “place”, but should have different Who's On First records, each with a unique and stable **wof:id** to represent each snapshot in time.

The rules in Who's On First may differ from the assumptions that a user or application has; documentation allows downstream data consumers and mapping services to optimize data usage and understand the assumptions in the data structure.

The folks over at the **UK's Local Ordnance Survey** (<https://www.ordnancesurvey.co.uk/>) wrote **a life cycle document** (<https://www.europa.uk.com/resources/os/os-mastermap-topography-layer-user-guide.pdf>) to detail a set of tracking rules around the features in their OS MasterMap database. The Local Ordnance Survey's User Guide was used as an outline for our own document due to it's complexity and detail. They did a great job at explaining the possible workflows around their feature IDs - a big thanks!

The following snippet from the Local Ordnance Survey document summarizes their reasonings well:

“The features within OS MasterMap vector layers are viewed as having a life cycle. The life cycle of each feature is matched, where practically possible, to that of the real-world object it represents. For example, a new building will become a new object in the Ordnance Survey main holding of the data and will be treated as the same feature – even if it undergoes change – until the building is demolished. By adopting this approach, Ordnance Survey is emulating real-world behaviour within a digital model and therefore creating a more realistic version of the real world in a computer.”

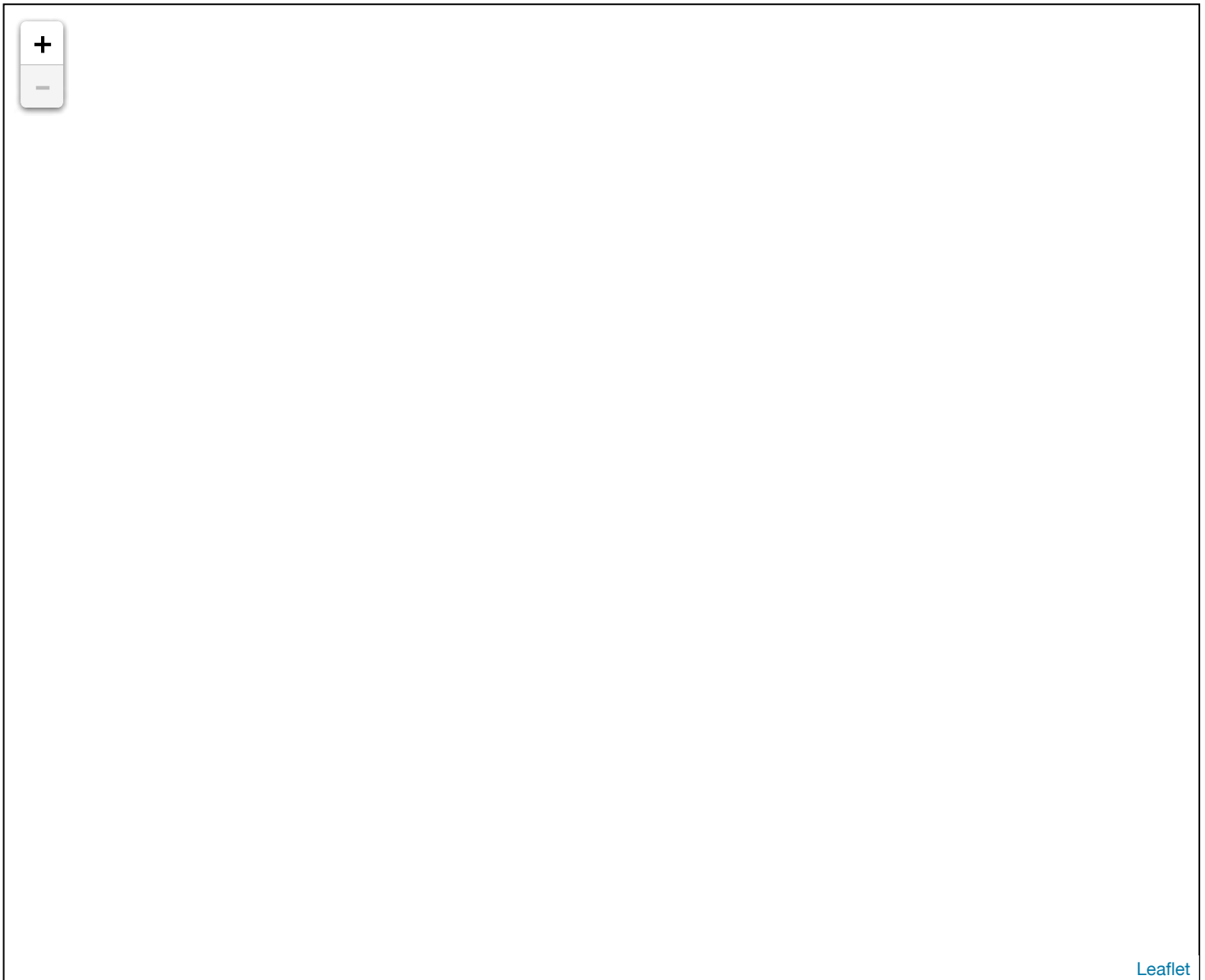
...

“Essentially, these rules indicate when an OS MasterMap feature will be retained and when it will be replaced, for different types of feature and different change scenarios. These rules are not only there to guide surveyors (from Ordnance Survey) collecting and attributing the features, but also to provide customers with a consistent definition of how real-world change is handled by Ordnance Survey.”

The Flowchart

And now, **our handy flowchart**

(https://raw.githubusercontent.com/whosonfirst/whosonfirst-cookbook/master/_images/lifecycle_flowchart.png) to help you visualize feature **wof:id** life cycle workflows in Who's On First:



The short version

- **Create:** This workflow should be followed for any **new** features that are being imported to Who's On First that the database does not already contain.
- **Alter:** Altered feature workflows occur to features that are **already** in Who's On First.

- **End of Life:** Used for features that are no longer current (think closed businesses or dissolved administrative districts).
- **Renew:** A workflow outcome that requires the existing feature to be “replaced” with a new feature. This requires some superseding work for the **wof:id**, see **our life cycle document** (https://github.com/whosonfirst/whosonfirst-cookbook/blob/master/definition/wof:id_lifecycle.md) for more detail.

Sounds great, but how does this work in practice?

In a **previous blog post** (<https://mapzen.com/blog/sf-neighbourhood-updates/>), we wrote about updating neighbourhood records in Who's On First. That neighbourhood work relied heavily on these life cycle rules because we not only created new neighbourhood records, but we also altered existing records and ended the life of several other records.

We liked what the UK's Ordnance Survey did, but we also recognize that Who's On First is a separate project with differing rules and guidelines. We thought about these rules and experimented; the neighbourhood work in San Francisco was the genesis of these life cycle rules. We've also begun updating neighbourhood records in other cities (New York, Los Angeles, Chicago, etc.), which rely on these same rules.

Let's take a look at use cases of the life cycle document through the eyes of the **Sunset District**; a neighbourhood in San Francisco that should *actually* be a macrohood.

Since Who's On First had a record for the Sunset District, this workflow will be an **alteration** to an existing place record. Let's walk through the workflow taken for the Sunset District:

Are we changing the feature's properties or geometry?

The feature's properties.

Was a change made to the wof:parent, wof:placetype, or wof:name fields?

Yes.

Was a change made to the wof:placetype?

Yes.

After following those steps in the flowchart, the outcome to the Sunset District record was to **renew**.

And there you have it. Since a change was made to the `wof:placetype` property of the existing Sunset District record, the record and its descendants had to be superseded into new records.

What about the Sunset District's descendant records?

The `wof:hierarchy` of the **Sunset Reservoir**, a venue that was parented by the existing Sunset District neighbourhood record, is shown below:

Before updating the Sunset District's record:

```
"wof:hierarchy": [
  {
    "continent_id":102191575,
    "country_id":85633793,
    "region_id":85688637,
    "county_id":102087579,
    "locality_id":85922583,
    "neighbourhood_id":85887463,
    "venue_id":588405947
  }
]
```

After updating the Sunset District's record:

```
"wof:hierarchy": [
  {
    "continent_id":102191575,
    "country_id":85633793,
    "region_id":85688637,
    "county_id":102087579,
    "locality_id":85922583,
    "macrohood_id":907212605,
    "neighbourhood_id":85865975,
    "venue_id":588405947
  }
]
```


Note the addition of a `marcohood_id` property and the change in the `neighbourhood_id` property value.

Internal debates and churn

It is important to note that the life cycle document, as written today, is a working document. We expect churn in the Who's On First database, which means we will not be able to capture every possible scenario or rule at the moment, but we are working towards being able to do just that.

While we do expect churn in the early days of Who's On First as we clean things up, we also expect things to stabilize going forward; this is just that nature of our work. We just need to remember that life is complicated and Who's On First mirrors those complexities.

Thanks for reading; please check out the full wof:id life cycle document **here** (https://github.com/whosonfirst/whosonfirst-cookbook/blob/master/definition/wof:id_lifecycle.md).

*If you or your application relies on Who's On First features, please give our document a read and **let us know what you think** (<mailto:stephen.epps@mapzen.com>)!*

Image via Dennis Yang, CC-BY 2.0 (<https://flic.kr/p/4NLV2d>)

· 06 October 2016 ·



Stephen Epps

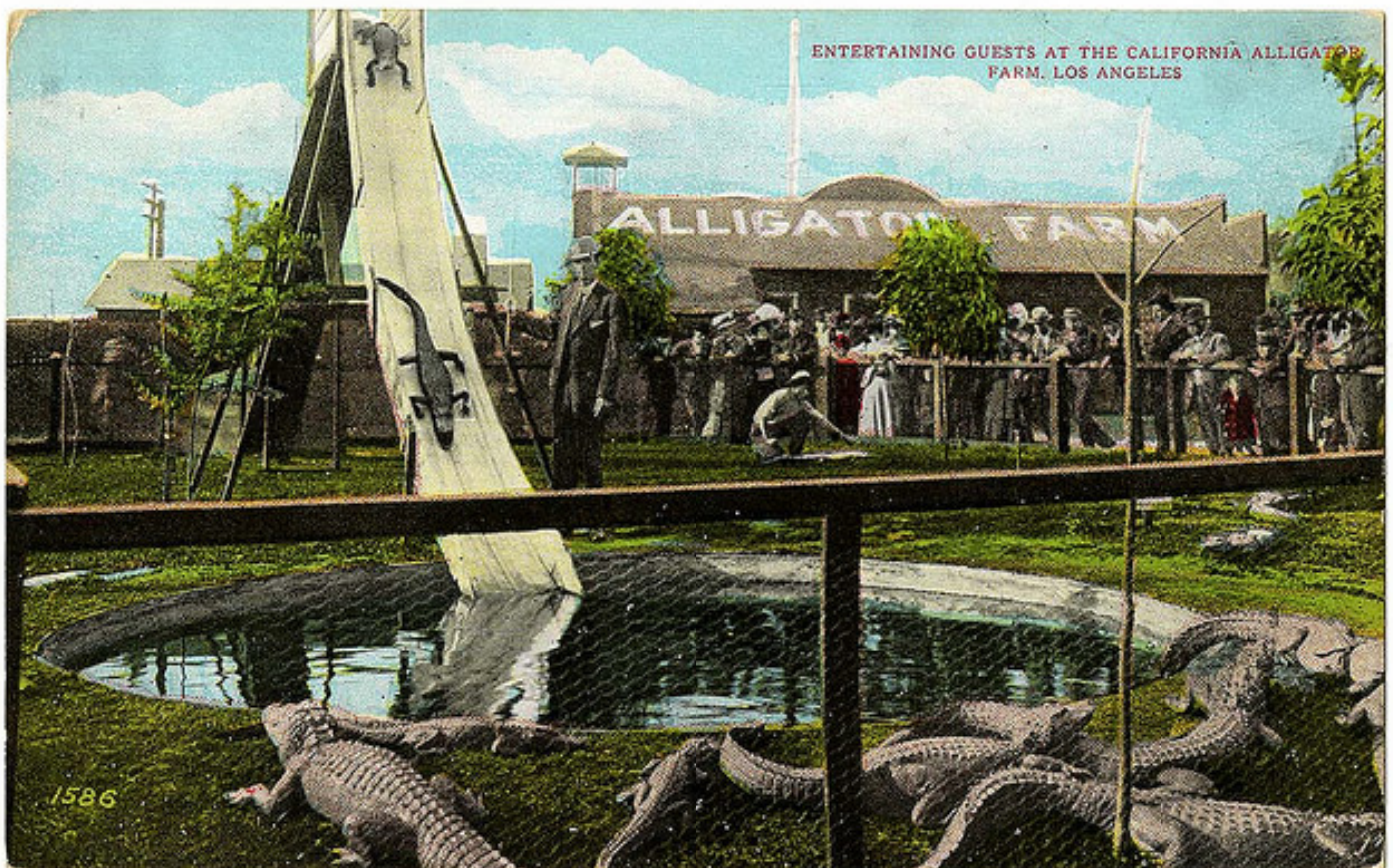
Stephen is Mapzen's gardener and fixer-upper of geographic data.

© 2017 Mapzen

Venues, Postal Codes... and All Those GitHub Repositories

whosonfirst (/tag/whosonfirst)

Venues



tl;dr - 20 million openly licensed venues, all with full (or at least partial) Who's On First hierarchies (<https://whosonfirst.mapzen.com/spelunker/placetypes/venue>).

What is this thing you call “venue”?

By default, Who's On First strives to group places in to **as few distinct place types as possible** (<https://github.com/whosonfirst/whosonfirst-placetypes>). The goal is to gather places in to buckets where they have *more in common with other kinds of places* than not. Initially we defined

venues as:

Things with four walls and a ceiling.

Then we remembered that we want to (eventually) add **all the landmarks in the world** (https://whosonfirst.mapzen.com/spelunker/tags/public_art/) to Who's On First. Rather than create a new placetype specific for landmarks we decided it made sense to classify them as venues, and to indicate their **landmark-iness** (<http://www.wikilovesmonuments.org/>) via a Mapzen-prefixed property.

Which meant we needed to change the definition of a venue to reflect many landmarks lack of walls or ceilings. Now we define a venue as:

Places that people might stand around, *together*.

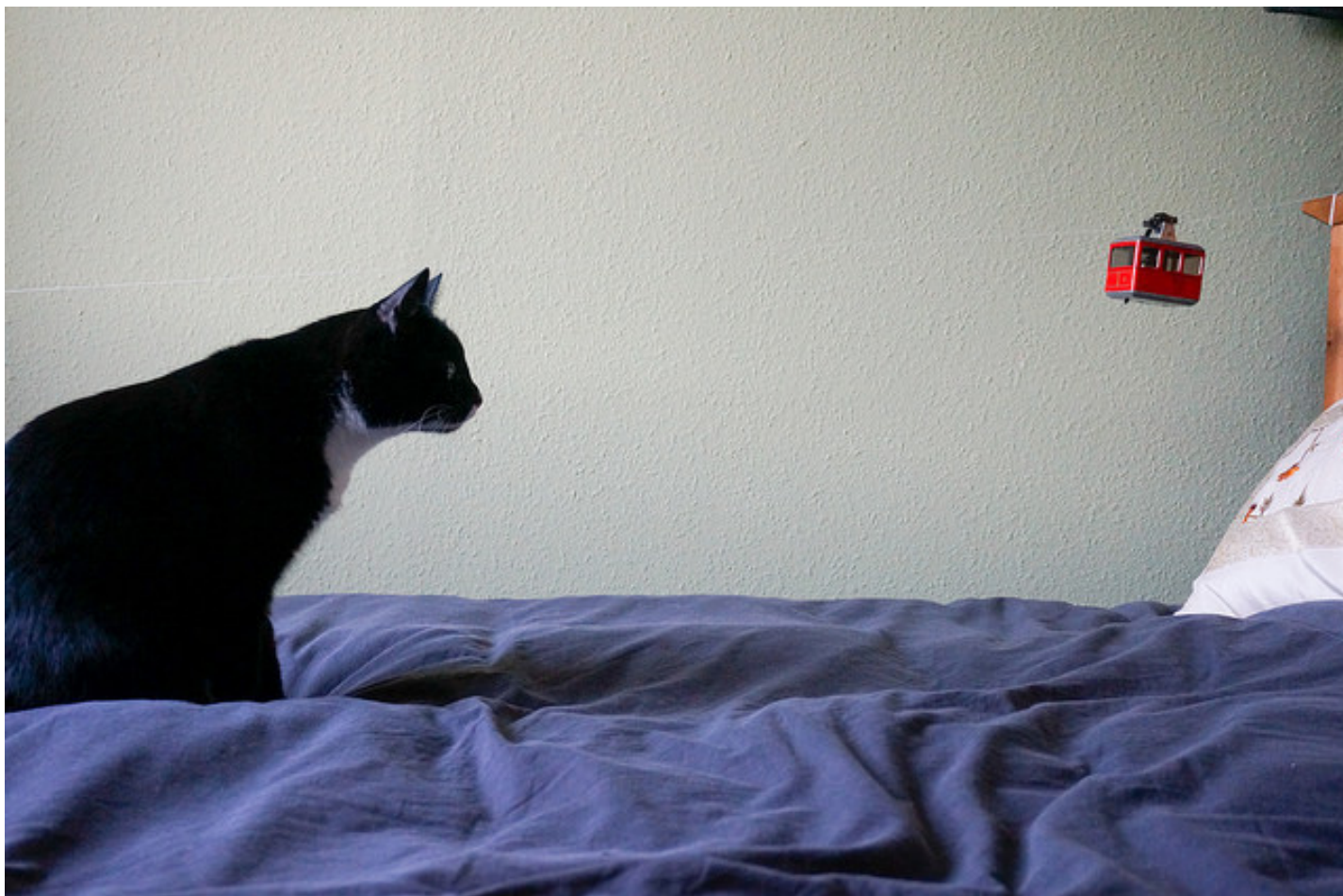
That may sound a little silly in the telling. You could find lots of fancier ways of saying the same thing but at the end of the day you'd still be... saying the same thing and in a way that would probably sound like gibberish to non-experts and strip away both the ambiguity and the play inherent in the reality of things.

So, that's what we think a venue **"is"** (https://www.youtube.com/watch?v=j4XT-l_3y0).

Venues are still the holy grail of geo data, precisely because they are the psychic underwear to our daily lives. For good or bad, most people's entire lives are mediated through "venues" be they commercial, cultural or social. There are a lot of venues, even in the smallest of communities.

Multiply *a lot of venues, even in the smallest of communities* by the *entire planet* and you've got... well, a lot of venues.

Which makes venues a hard problem. It's a hard problem collecting all those venues. It's a hard problem vetting them and to keeping them up to date. It's a hard problem to figure out where to *put* them all and harder still to figure out how to make that many things searchable. These are hard problems and that has meant only companies with an interest in reselling access to that data have endeavoured to take up the challenge. That's not very interesting to Who's On First.



Historically, there haven't been a whole lot venue databases that don't impose constraints, of one kind or another, on their re-use. The goal for Who's On First is to have an openly licensed database of places that *may* be used in a commercial project, without any additional restrictions. Something akin to an **MIT** (https://en.wikipedia.org/wiki/MIT_License) or **BSD** (https://en.wikipedia.org/wiki/BSD_licenses) software license but for data. That is not everyone's definition of "open" and that's okay. There are different flavours of "open" and different reasons for choosing one over another.

Who's On First **has always been a wildly ambitious project** (<https://mapzen.com/blog/mapping-with-bias/>) so rather than be discouraged by something as big and hairy and complicated as venues, we chose instead to jump in with both feet.

SimpleGeo (2009–2011)

In 2011, the now defunct geo-services company **SimpleGeo** (<https://www.twitter.com/simplegeo>) published their **Public Spaces Collection** (<https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces>), 20 million business listings, as a **Creative Commons Zero**

(<https://creativecommons.org/publicdomain/zero/1.0/>) (CC0) dataset. **Thanks, SimpleGeo!** (<https://web.archive.org/web/20110424090705/http://blog.simplegeo.com/2011/04/20/open-places-data/>)

Most venues are in the **USA** (<https://whosonfirst.mapzen.com/spelunker/placetype/venue?iso=us>) (approximately 12 million of them) but other countries with a not-insignificant number of venues include the **UK** (<https://whosonfirst.mapzen.com/spelunker/placetype/venue?iso=gb>), **Germany** (<https://whosonfirst.mapzen.com/spelunker/placetype/venue?iso=de>), **Canada** (<https://whosonfirst.mapzen.com/spelunker/placetype/venue?iso=ca>), **Spain** (<https://whosonfirst.mapzen.com/spelunker/placetype/venue?iso=es>) and **Australia** (<https://whosonfirst.mapzen.com/spelunker/placetype/venue?iso=au>). There are still *many, many* places in the world where the SimpleGeo dataset doesn't have any venue data so this is a just a beginning, and not a triumphal celebration on arrival. You can see a complete count of venues by country here:

<https://github.com/whosonfirst-data/whosonfirst-data-venue/blob/master/DATA.md>
(<https://github.com/whosonfirst-data/whosonfirst-data-venue/blob/master/DATA.md>)

In 2015, we started importing those venues in to Who's On First. The import process has been a start-and-stop affair and adding a little over 8 million venues, last year, we put things on hold and didn't pick them up again until earlier this year. That work **wrapped**

(<https://github.com/whosonfirst-data/whosonfirst-data-venue-us/issues/1>) up (<https://github.com/whosonfirst-data/whosonfirst-data-venue/issues/1>), recently, meaning every one of those 20 millions venues has been assigned **their very own Who's On First ID** (<https://mapzen.com/blog/wof-lifecycle-documentation/>) and a full set of ancestors within **Who's On First** (<https://github.com/whosonfirst/whosonfirst-placetypes#here-is-a-pretty-picture>).

Aside from a general desire to add all those venues in to Who's On First the sheer volume of data was attractive as a way to test some of the assumptions and infrastructure we've developed to manage **all these places** (<https://mapzen.com/blog/who-s-on-first/>) we've taken under our wing. Any comprehensive database of venues in Who's On First, whether we seeded it using SimpleGeo or another source, would have all the same problems of scale so even though there were some initial concerns about the quality and the focus of the Public Spaces Collection we figured "Why not at least use it to start testing things, now?"



The SimpleGeo data was released in 2011 and it's 2016, today, so it's safe to assume that we will *not* have a venue record for that cool new bar which just opened in **London** (<https://whosonfirst.mapzen.com/spelunker/ID/descendants&placetype=venue>) or **Los Angeles** (<https://whosonfirst.mapzen.com/spelunker/ID/descendants&placetype=venue>) or **Shanghai** (<https://whosonfirst.mapzen.com/spelunker/ID/descendants&placetype=venue>). We aspire to have those venues, and aspire to have them in something approaching real-time, but today we do not.

*We might actually have some of those cool new bars in New York City or San Francisco but that is largely a function of choosing those two cities to start testing **the Who's On First editorial stack** (<https://mapzen.com/blog/boundary-issues-properties>) with, since that is where most of the Mapzen staff live. This coastal bias is not a feature of Who's On First, only reflective of the fact that it is still early days...*

On the other hand if you need a plumber or a notary in any of those cities, or places like rural Kansas, it's entirely possible **we have those listings** (https://whosonfirst.mapzen.com/spelunker/search/?q=plumber®ion_id=85688555).

That's the interesting thing about the term "venues", in 2016: It has become short-hand for a very specific *subset* of venues, eclipsing everything else. Often when people say "venues" they're really talking about restaurants and bars in urban or metropolitan areas and shops where people arguably enjoy spending money on the things they're buying (as opposed to, say, picking up toilet paper at the drug store).

These venues are also almost always *current* venues suggesting the thrill, however illusory, of something that **hasn't been discovered by everyone else yet** (<http://www.aaronland.info/weblog/2014/10/06/interpretation/#brick>) and aimed very specifically at what marketers refer to as "the 18-35 year old" demographic.



But let's stop for a moment and remember a few things: Let's remember the **nearly 100-year old butcher shop** (<https://whosonfirst.mapzen.com/spelunker/id/572151977/>) in your neighbourhood. Or the **local bar** (<https://whosonfirst.mapzen.com/spelunker/id/572077043/>) that closed last year **whose history runs deeper** (https://en.wikipedia.org/wiki/The_Lexington_Club) than a pint glass. Let's remember our new best friend **Poop Emoji Rock** (<https://whosonfirst.mapzen.com/spelunker/id/1008184051/>).

As important as it is, there is more to life than tomorrow's happy hour. Plus, sooner or later, everyone gets ejected from **18-35 Year Old Club** (<http://www.aaronland.net/old-club/>) and then what?

As we've been working with the SimpleGeo dataset it's become clear that despite some bunk data and some unfortunate absences if there was a venue that existed before 2011 which had any kind of following, or traffic, it's probably in there... *somewhere*.

Now that we've gotten over the initial hurdle of simply importing all those venues (more on that below) a big part of our work in the short-term is to think about how we can allow people to make sense of all that stuff. It's easy to complain that there are no open venues in the world but then to be confronted by **86, 000 accountants in seven different countries** (<https://whosonfirst.mapzen.com/spelunker/placetypes/venue/?tag=accountant>) can be equally disorienting.

Some of the work going forward will involve **Yes-No-Fix style tools** (<https://mapzen.com/blog/yesnofix>) allowing people to quickly flag venues as being still-open or long-since-closed and some of the work will involve **grouping venues in to categories** (<https://github.com/whosonfirst/whosonfirst-categories>) for directed searches and general browsing. Some of the work will simply be learning by building things and seeing what works and **what doesn't** (<https://www.youtube.com/watch?v=XnoqLtJbjcl>).

It's not going to be perfect right away but if we do the job well then, at the very least, things will be *better than they were yesterday*.

In the meantime there are 20, 848, 132 (and counting) venues that have been **Who's On First - ifed** (<https://whosonfirst.mapzen.com/spelunker/placetypes/venue>).

Postal Codes



tl;dr - **3 million openly licensed postal codes with centroids or polygons**
(<https://whosonfirst.mapzen.com/spelunker/placetypes/postalcode/?exclude=nullisland>)
in 14 countries.

In the same spirit as venues we've also started to tackle adding postal codes to Who's On First. The initial import of postal codes came courtesy the 7.10.0 release of the **Yahoo! GeoPlanet dataset** (https://archive.org/details/geoplanet_data_7.10.0.zip), which contains postal codes for most countries, even if they don't include geometries. That's okay because each postal code does have a **parent ID** (<https://whosonfirst.mapzen.com/spelunker/concordances/geoplanet/>) and that gives us enough information to group things by country and we can add geometries, opportunistically, one country at a time.

There are more than 14 countries in the world so there is a lot of work left to do but here are some of the highlights, to date:

Thank you, Ordnance Survey!

We've imported all of the UK Ordnance Survey's **Code-Point Open** (<https://www.ordnancesurvey.co.uk/business-and-government/products/code-point-open.html>) dataset in to Who's On First and we're still trying to sort out why there are a healthy 300, 000 UK postal codes *without* geometries. On the other hand there *are* **1.6 million postal codes with centroids** (<https://whosonfirst.mapzen.com/spelunker/placetypes/postalcode/?iso=gb&exclude=nullisland>) so that's progress.

Thank you, Hannes and Kevin!

Thanks to **Hannes Junnila** (<https://github.com/hannesj>) and Mapzen's own **Kevin Kreiser** (<https://twitter.com/kevinkreiser>) we have imported official postal codes for **Finland** (<https://github.com/whosonfirst-data/whosonfirst-data-postalcode-fi/issues/1>) and **Austria** (<https://github.com/whosonfirst-data/whosonfirst-data-postalcode-at/issues/1>) and **Switzerland** (<https://github.com/whosonfirst-data/whosonfirst-data-postalcode-ch/issues/1>).

Thank you, America and Canada!

Likewise we've imported the **US Census ZIP Code Tabulation Areas** (<https://www.census.gov/geo/reference/zctas.html>) meaning we have geometries for around **75% of all the postal codes** (<https://whosonfirst.mapzen.com/spelunker/placetypes/postalcode/?iso=us&exclude=nullisland>) in the USA.

North of the US border, it turns out that Canada has about **800, 000 six-character postal codes** (<https://whosonfirst.mapzen.com/spelunker/placetypes/postalcode/?iso=ca>). The licensing around their geometries **has a long and tortured history** (<http://www.michaelgeist.ca/2016/06/crowdsourcedpostalcelawsuit/>). Statistics Canada, however, has been nice enough to **publish shapefiles for the 1, 621 Forward Sortation Areas (FSA)** (<http://www12.statcan.gc.ca/census-recensement/2011/geo/bound-limit/bound-limit-2011-eng.cfm>) which are represented by the first three characters in a Canadian postal code.

Using that information we generated **approximate centroids for the all the six-character postal codes** (<https://whosonfirst.mapzen.com/spelunker/placetypes/postalcode/?iso=ca&exclude=nullisland>) derived from the geometric center of their parent FSA. This is not Perfect Data but it does scope the problem to about a square kilometer rather than an **entire province** (https://en.wikipedia.org/wiki/Postal_codes_in_Canada).

Thank you, Open Addresses!

Using the **Clustr** (<https://github.com/whosonfirst/Clustr>) tool originally developed to **extract shapefiles from geotagged Flickr photos** (<http://code.flickr.net/2008/10/30/the-shape-of-alpha/>) we've been able to generate approximate geometries for postal codes derived from **Open Addresses** (<http://openaddresses.io/>) data for **Australia** (<https://github.com/whosonfirst-data/whosonfirst-data-postalcode-au/>), the **Netherlands** (<https://github.com/whosonfirst-data/whosonfirst-data-postalcode-nl/>) and **France** (<https://github.com/whosonfirst-data/whosonfirst-data-postalcode-fr/>).

These geometries are very much Weird Data so it is left up to you to decide whether you think they're better than no data at all. If nothing else we think having to option to make that choice is better than no data.

All Those GitHub Repositories



tl;dr - We are betting on the future and **making do with the present** (<https://github.com/whosonfirst-data>).

Recently we created a second organization in GitHub for Who's On First related work. The first organization is called **whosonfirst** (<https://github.com/whosonfirst>) and houses all of the software we've written to date as well as a growing body of theory and **documentation** (<https://github.com/whosonfirst/whosonfirst-cookbook>).

The second organization is called **whosonfirst-data** (<https://github.com/whosonfirst-data>) and it is where the millions and millions of GeoJSON files representing **all of the places** (<https://mapzen.com/blog/all-of-the-places/>) in Who's On First live. The decision for a second organization was spurred on by the work we've been doing to import venues and postal codes.

Work that has translated in to *24 million GeoJSON files spread across 488 GitHub repositories*. Before we go any further, let's be clear about one thing: This is not an ideal situation.

Ultimately our goal is to have a single monolithic Who's On First repository that will contain *all 24 million* (and counting) records. In 2016 storing 24 million tiny files in a single Git repository is either technically impossible or so impractical as to "play impossible on TV".

Until that better day comes when a single "mono-repo" is possible we have been working instead to establish conventions for what repository a given file lives in and what that repository is called. The naming conventions for repositories at their *most granular* are as follows:

"whosonfirst-data-" + WHOSONFIRST_PLACETYPE + "-" + WHOSONFIRST_COUNTRY_CODE + "-" + WHOSONFIRST_SUBDIVISION_CODE

For example:

- **whosonfirst-data** (<https://github.com/whosonfirst-data/whosonfirst-data>) — administrative data (**continents - microhoods** (<https://github.com/whosonfirst/whosonfirst-placetypes>)) for the world
- **whosonfirst-data-venue-us-ca** (<https://github.com/whosonfirst-data/whosonfirst-data-venue-us-ca>) — venues in California, USA
- **whosonfirst-data-venue-ca** (<https://github.com/whosonfirst-data/whosonfirst-venue-ca>) — venues in Canada
- **whosonfirst-data-postalcode-fi** (<https://github.com/whosonfirst-data/whosonfirst-data-postalcode-fi>) — postalcodes in Finland

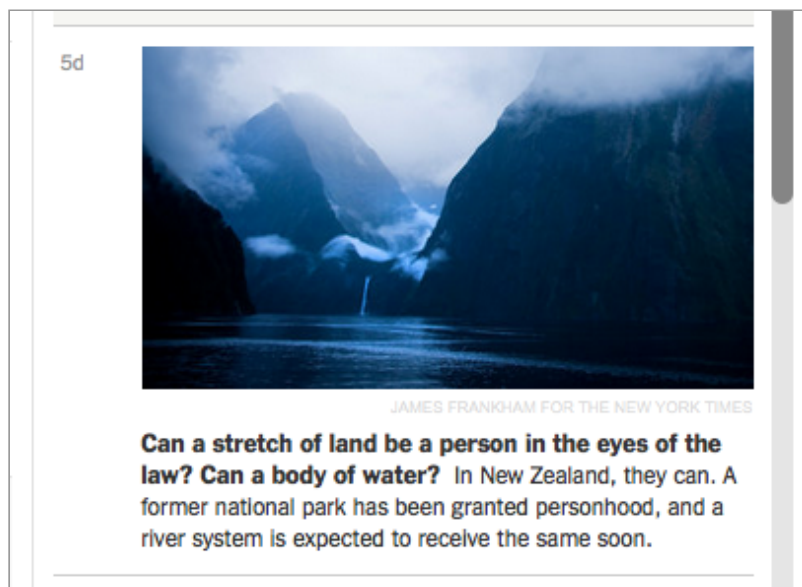
The first thing to note is that not all repositories are as granular as the rules described above.

Wherever feasible we try to bundle records with the *least amount* of granularity as possible. For example postalcodes are grouped by country as are venues unless there are so many of them, like in the USA, that it is not practical to keep them in a single parent repository.

If a repository grows so much data that it is no longer practical to keep everything in one place then it may be subdivided in to a number of child repositories. Venues are a good example of this.

We try to maintain a separate “parent” repository for things that have been broken out in to multiple child repositories. For example there is a **whosonfirst-data-postalcode** (<https://github.com/whosonfirst-data/whosonfirst-data-postalcode>) repository that contains no data but instead a **pointer** (<https://github.com/whosonfirst-data/whosonfirst-data-postalcode/blob/master/data.json>) to all the repositories that do have postalcode data. We also do the same for **venues in the USA** (<https://github.com/whosonfirst-data/whosonfirst-data-venue-us>).

The **whosonfirst-data** (<https://github.com/whosonfirst-data/whosonfirst-data>) repository is the obvious exception (or perfect example, depending on how you look at it) to the scenario described above. This repository contains all “administrative” placetypes (all the places between and inclusive of continents to microhoods) for the entire world. While it is possible to imagine that the sum total of all the neighbourhoods in the world will require putting them in a separate repository but we are going to hold off doing that for as long as we can.



(<http://www.nytimes.com/2016/07/14/world/what-in-the-world/in-new-zealand-lands-and-rivers-can-be-people-legally-speaking.html>)

This practice is still in its early stages so we apologize in advance if it's bumpy, weird or incomplete.

The goal is to provide a clear and reproducible template for subdividing a repository (that has grown too large) in such a way that all those repositories may eventually be merged in to a single collection without any conflicts. Who's On First documents are grouped by country and placetype as a convenience but, technically, they can live anywhere.

Finally, Who's On First records should always have a `wof:repo` property indicating the repository to which they belong. If they don't that's a bug.

Huzzah!

Images used in this blog post:

- **Entertaining guests at the California alligator farm, Los Angeles**
(https://www.flickr.com/photos/chs_commons/16356193176/) *California Historical Society*
- **Franzi and cable car** (<https://www.flickr.com/photos/philgyford/14251596747/>) *Phil Gyford*
- **Postcard of the sea serpent** (<https://www.flickr.com/photos/nantuckethistoricalassociation/3177485572/>)
Nantucket Historical Association
- **Turtle club, caught between San Pedro & Catalina Island, Cal.**
(https://www.flickr.com/photos/chs_commons/15485394421/) *California Historical Society*
- **Sitting with travel trailer at St. Petersburg Beach**
(<https://www.flickr.com/photos/floridamemory/14889402267/>) *State Library and Archives of Florida*
- **The Endless Staircase** (<https://www.flickr.com/photos/benterrett/9785396672/>) *Ben Terrett*

· 07 October 2016 ·



Aaron Straup Cope

Aaron is happiest in the presence of olive oil.

© 2017 Mapzen

Mapzen Terrain Tiles are 1.0 and ready to go

[tangram \(/tag/tangram\)](/tag/tangram) [data \(/tag/data\)](/tag/data) [terrain \(/tag/terrain\)](/tag/terrain)

[documentation \(/tag/documentation\)](/tag/documentation) [vector-tiles \(/tag/vector-tiles\)](/tag/vector-tiles)

We're pleased to announce **worldwide elevation data** by graduating our terrain-preview project into a fully supported Mapzen Terrain Tiles service!

Terrain tiles provide open elevation data for the entire world in a raster tile format. By aggregating, standardizing, and tiling multiple source datasets into common web mapping and desktop GIS formats and dimensions, it's much easier to process the data and build applications on top of it.

Each square-shaped terrain tile contains a grid of elevation values. For example, the grid cell for Mount Everest would have a value of 8,848 meters above sea level, while the grid cell for Challenger Deep has a value of 10,994 meters below sea level.

With terrain tiles, you have the power to customize the content and visual appearance of your map on the fly and perform complicated analysis on the desktop or in the cloud. We're excited to see what you build!



(Open full screen ↗ (<http://tangrams.github.io/terrain-demos?url=styles/imhof.yaml#12/37.8773/-121.9290>))

If you are familiar with digital elevation models (DEMs) or digital terrain models (DTMs), then you'll be interested in Mapzen's new Terrain Tiles service. Tiles are available for zooms 0 through 15 in several spatial data formats, including PNG and GeoTIFF tiles in Web Mercator projection, as well as SRTM-style HGT in raw latlng projection.

A derived "normal map" product is also available. We precompute the math on the server so renderers like **Tangram** (<https://mapzen.com/products/tangram/>) can generate spheremap-based hillshades and real-time 3D scenes with these "normal" tiles, even on your phone.

Get an API key

To use the Mapzen Terrain Tiles service, you should first get a developer API key. Sign in at **<https://mapzen.com/developers>** (<https://mapzen.com/developers>) to create and manage your API keys, it'll only take a minute (or two).

1. Go to **<https://mapzen.com/developers>** (<https://mapzen.com/developers>).

2. Sign in with your GitHub account. If you have not done this before, you need to agree to the terms first.
3. Create a new key, and optionally, give it a name so you can remember the purpose of the project.
4. Copy the key into the terrain URL where `mapzen-xxxxxxx` represents your key.

Tell me more

Complete documentation for the Mapzen Terrain Tiles service is available from:

<https://mapzen.com/documentation/terrain-tiles/>
(<https://mapzen.com/documentation/terrain-tiles/>)

Endpoints

Mapzen terrain tile endpoints are:

- `https://tile.mapzen.com/mapzen/terrain/v1/terrarium/{z}/{x}/{y}.png?api_key=mapzen-xxxxxxx`
- `https://tile.mapzen.com/mapzen/terrain/v1/normal/{z}/{x}/{y}.png?api_key=mapzen-xxxxxxx`
- `https://tile.mapzen.com/mapzen/terrain/v1/geotiff/{z}/{x}/{y}.tif?api_key=mapzen-xxxxxxx *`
- `https://tile.mapzen.com/mapzen/terrain/v1/skadi/{N|S}{y}/{N|S}{y}{E|W}{x}.hgt.gz?api_key=mapzen-xxxxxxx`

The earlier `terrain-preview.mapzen.com` endpoints announced March 22nd, 2016 will be retired effective **January 1st, 2017**. Please update your projects.

(*) Note: GeoTIFF format tiles are 512x512 sized so request the parent tile's coordinate. For instance, if you're looking for a zoom 14 tile then request the parent tile at zoom 13.

Additional Amazon Public Dataset Endpoints

We're proud to partner with Amazon to offer terrain tiles in their **public dataset** (<https://aws.amazon.com/public-data-sets/terrain/>) program on S3.

If you're building in Amazon AWS we recommend using machines in the **us-east** region (the same region as the S3 bucket) and use the following endpoints for increased performance:

- `https://s3.amazonaws.com/elevation-tiles-prod/terrarium/{z}/{x}/{y}.png`

- <https://s3.amazonaws.com/elevation-tiles-prod/normal/{z}/{x}/{y}.png>
- <https://s3.amazonaws.com/elevation-tiles-prod/geotiff/{z}/{x}/{y}.tif>
- <https://s3.amazonaws.com/elevation-tiles-prod/skadi/{N|S}{y}/{N|S}{y}{E|W}{x}.hgt.gz>

File formats

- **Terrarium** format PNG tiles contain raw elevation data in meters, in Mercator projection (EPSG:3857). All values are positive with a 32,768 offset, split into the red, green, and blue channels, with 16 bits of integer and 8 bits of fraction. To decode:

$$(\text{red} * 256 + \text{green} + \text{blue} / 256) - 32768$$

- **Normal** format PNG tiles are processed elevation data with the the red, green, and blue values corresponding to the direction the pixel “surface” is facing (its XYZ vector), in Mercator projection (EPSG:3857). The alpha channel contains quantized elevation data with values suitable for common hypsometric tint ranges. High alpha channel values indicate lower elevation values (below sea level), making them more opaque.

Specifically, normal format alpha values are counted in (floored) elevation increments. Below sea level they start at -11,000 meters (Mariana Trench) and range to -1,000 meters in 1,000 meter increments, with more detail on the coastal shelf at -100, -50, -20, -10 and -1 meters and finally 0 (intertidal zone). Values above sea level are reported in 20 meter increments to 3,000 meters, then 50 meter increments until 6,000 meters, and then 100 meter increments until 8,900 meters (Mount Everest).

- **GeoTIFF** format tiles are raw elevation data suitable for analytical use and are optimized to reduce transfer costs in 512x512 tile sizes but with internal 256x256 image pyramiding, in Mercator projection (EPSG:3857). Allow for the larger tile size by referring to the tile coordinate of {z-1} parent tile.
- **Skadi** format tiles are raw elevation data in unprojected latlng (EPSG:4326) 1°x1° tiles, used by the Mapzen Elevation Service. Essentially they are just the SRTMGL1 format tiles but with global coverage. See **the SRTM guide** (https://lpdaac.usgs.gov/sites/default/files/public/measures/docs/NASA_SRTM_V3.pdf) for exact format specifications.

Data sources

Mapzen aggregates elevation data from several open data providers including 3 meter and 10 meter **3DEP** (<http://nationalmap.gov/elevation.html>) in the United States, 30 meter **SRTM** (<https://lta.cr.usgs.gov/SRTM>) globally, and coarser **GMTED** (http://topotools.cr.usgs.gov/gmted_viewer/) zoomed out and **ETOPO1** (<https://www.ngdc.noaa.gov/mgg/global/global.html>) to fill in bathymetry, all powered by GDAL/OGR **VRTs** (http://www.gdal.org/drv_vrt.html) and some special sauce.

Could we provide better coverage in your area? Please recommend additional open datasets to include by **filing an issue** (<http://github.com/mapzen/joerd/issues/new>).

Related services

- **Elevation query service:** To query elevation at a single point or along a path almost anywhere in the globe, **sign up for an API key** (<https://mapzen.com/developers/>) and check out the **Mapzen Elevation Service** (<https://mapzen.com/documentation/elevation/>).
- **Elevation influenced bicycle routing:** <https://mapzen.com/blog/making-the-grade-elevation-influenced-bicycle-routing/> (<https://mapzen.com/blog/making-the-grade-elevation-influenced-bicycle-routing/>)
- **Walkabout outdoor map:** <https://mapzen.com/blog/walkabout/> (<https://mapzen.com/blog/walkabout/>)

Terrain timeline

- Back in March we previewed tiles in California with Peter's **Mapping Mountains** (<https://mapzen.com/blog/mapping-mountains/>) post using a pre-release version of Tangram to generating hillshades client-side.
- We followed up with May's **What a Relief: Global Test Pilots Wanted** (<https://mapzen.com/blog/elevation/>) letting you know about improved detail in the United States and official Tangram **support** (<https://github.com/tangrams/tangram/releases/tag/v0.7.0>).
- In early July we launched **Walkabout** (<https://mapzen.com/blog/walkabout/>), Mapzen's signature outdoor cartography.
- In late July Geraldine explored the world of **Sphere Maps** (<https://mapzen.com/blog/sphere-maps/>) based on the terrain data.
- In September Peter followed up again with a new **Heightmapper** (<https://mapzen.com/blog/tangram-heightmapper/>) tool to browse the terrain tiles and create heightmaps.
- Later in September we partnered with Amazon to release the raw data as new **terrain public dataset** (<https://aws.amazon.com/public-data-sets/terrain/>).

- In October (that's now!) we're releasing v1 of the Mapzen Terrain Tiles service.
- NOTE: The existing terrain-preview.mapzen.com endpoints will be retired Jan. 1, 2017.

Feedback

Have any feedback on the terrain tiles? Please direct feedback to **Nathaniel Vaughn Kelso** (<https://github.com/nvkelso>), our Chief Cartographer, at **tiles@mapzen.com** (<mailto:tiles@mapzen.com>)!

· 11 October 2016 ·



Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.



Matt Amos

OpenStreetMap developer-contributor and chilli enthusiast. Writes vector tile and other data-mastication tools at Mapzen.

© 2017 Mapzen

Let's get cartographical at NACIS 2016!

vector-tiles ([/tag/vector-tiles](#)) **whosonfirst** ([/tag/whosonfirst](#)) **data** ([/tag/data](#))

Fall always brings so many great seasonal offerings: apple-picking, beautiful foliage, and the annual **North American Cartographic Information Society (NACIS) conference** (<https://nacis.org/2016>)! With the help of our Walkabout basemap, we're excited to make the hike to Colorado Springs for NACIS to talk about maps, data, and terrain.



Close Controls

[Leaflet](#)

Colorado Springs, here we come! View full screen ↗ (<https://tangrams.github.io/walkabout-style-more-labels/#12.0566/38.8516/-104.8497>)

We'll hear about **Vector Tiles** (<https://mapzen.com/projects/vector-tiles/>) from **Katie** (<https://twitter.com/KatieKowalsky>) and our **Who's On First** (<https://whosonfirst.mapzen.com/>) gazetteer from **Nathaniel** (<https://twitter.com/kelsosCorner>). **Ekta** (<https://twitter.com/ektadary>) and **Peter** (<https://twitter.com/meetar>) will also be at NACIS, ready to chat about maps and share some of the finest geo stickers.

Here's a rundown of all the talks our team will be giving:

- **Breaking up with Raster and Going Steady with Vector Tiles** (<https://nacis2016.sched.org/event/7JLa/practical-cartography-day-early-morning-session>) (Practical Cartography Day, 9:00) - *Katie Kowalsky*
- **Who's on First: Administrative Boundaries and Localities** (<https://nacis2016.sched.org/event/7Lj4/drawing-the-line>) (Thursday, 9:00) - *Martin Gamache, Nathaniel Vaughn Kelso*
- **Who ARE the People in your Neighborhood? Developing Mapzen's Neighborhood Database** (<https://nacis2016.sched.org/event/7Lj4/drawing-the-line>) (Thursday, 9:00) - *Nat Case, Nathaniel Vaughn Kelso*

· 12 October 2016 ·



Katie Kowalsky

Katie is a mapslinger who knows how to make a proper cheese plate and is a serial to-do list maker.

© 2017 Mapzen

The Next Chapter of Search

geocoding (/tag/geocoding)

We're shaking things up a bit here in Search land! We recently released Pelias backed by libpostal for analyzing a `/v1/search` input into its constituent address parts coupled with taking steps to ensuring that only the most accurate results are returned. This is the first release of many to fulfill our goal of turning Pelias (and thereby Mapzen Search) into a top tier geocoder and POI search engine.

As Mapzen Search is the service and Pelias is the actual software that powers it, any changes made to Pelias are reflected in the user experience of the Mapzen Search API.



image **via Cooper Hewitt** (<https://collection.cooperhewitt.org/objects/18392723/>), Sidewall, French Village, 1950–52; Manufactured by Katzenbach and Warren, Inc.

Libpostal

As described in this **blog post** (<https://mapzen.com/blog/inside-libpostal/>) by **Al Barrentine** (<https://github.com/thatdatabaseguy>), libpostal is an international address parser trained on **OpenStreetMap** (<https://www.openstreetmap.org/>), one of the primary sources for venue and address data in Mapzen Search. By incorporating libpostal, we're moving away from **addressit** (<https://github.com/DamonOehlman/addressit>), a module that specializes in identifying the street tokens of an input while leaving the remaining administrative area and query tokens loosely identified. This has worked pretty well for Mapzen Search so far but we've had to jump through some hoops to incorporate the output intelligently into subsequent steps of the API pipeline. Now, as we transition to more of a true geocoder with business rules beyond term matching, it makes sense to switch to libpostal.

As libpostal is a full address parser, its usage only applies to input into the `/search` endpoint. libpostal is not used in the `/autocomplete`, `/place`, and `/nearby` endpoints.

Only Return What You Asked For

Evolving how a production geocoder operates is tricky business. There are lots of edge cases to consider but fortunately we can change incrementally. This will be the first step in a long journey towards geocoding enlightenment that we're confident will make a better experience for our users.

As this section is entitled, our driving philosophy going forward will be to only return results that apply to what you asked for. For example, consider a search for `30 W 26th Street, New York, NY`. Until the introduction of libpostal, this is what Mapzen Search returned:

- 1) 30 West 26th Street, Manhattan, New York, NY, USA
- 2) 30 West 26th Street, Bayonne, NJ, USA
- 3) 30 West 26th Street, Millcreek, PA, USA
- 4) 30 West 26th Street, Merced, CA, USA
- 5) 30 West 26 Street, Manhattan, New York, NY, USA
- 6) 30 West 26 Street, Manhattan, New York, NY, USA
- 7) 253 West 26th Street, Manhattan, New York, NY, USA
- 8) 501 West 26th Street, Manhattan, New York, NY, USA
- 9) 455 West 26th Street, Manhattan, New York, NY, USA
- 10) 171 West 26th Street, Manhattan, New York, NY, USA

The results consisted of a very precise result first since it matched the most tokens along with results that matched *most* but not all of the tokens. Note that in the 2nd-4th results, the house number and street matched while in the 7th-10th results, the street, city, and state matched while the house number is different. This is because the geocoder gave a lot of weight to the number of tokens matched without determining whether those results made sense.

With the libpostal integration and introduction of geocoding rules, the geocoder will now return only the most accurate result:

1) 30 West 26th Street, Manhattan, New York, NY, USA

How It Works

The input is first passed to libpostal for text analysis. If the parsed input has a street then Elasticsearch is queried for the parsed input, otherwise the geocoder will use a geographic search engine approach to resolving the input. Take, for example, 30 West 26th Street, New York, NY . Libpostal parses this as:

```
{
  "house_number": "30",
  "road": "west 26th street",
  "city": "new york",
  "state": "ny"
}
```

Much like before, the geocoder will query Elasticsearch for the parsed input fields, though libpostal more accurately identifies the non-street portions of the input. The difference is now that the geocoder will only return results that match **all** of the parsed fields that are queried for. As illustrated above, this is a departure from subjecting the input to a geographic search engine approach.

What happens when user intent fails?

In an ideal world, the data would be perfect and users would only enter inputs that are perfectly formed, spelled correctly, and geographically make sense. Here are some scenarios in which an input can be incorrect:

- misspelling, e.g. - Appalloosa Way instead of Appaloosa Way
- street name changes between towns, e.g. - West Broadway turns into East Main Street
- a street name may not exist in a city, e.g. - 1220 Calle de Lago, New York, NY

Libpostal will happily parse these inputs as containing a street because they are all technically valid, just not geographically consistent:

```
{
  "house_number": "1220",
  "road": "calle de lago",
  "city": "new york",
  "state": "ny"
}
```

1220 Calle de Lago is a valid address in Socorro, NM (among other cities) but not in New York City.

To deal with these problematic inputs, the query logic now discards the most granular parts of the parsed input to attempt to match something less granular. In this case, 1220 Calle de Lago isn't in New York, NY, so it will discard the house number and street and attempt to just match New York, NY .

First, let's look at an input that matches everything: 30 West 26th Street, Manhattan, New York, NY, USA , which libpostal would interpret as:


```
{
  "house_number": "30",
  "road": "west 26th street",
  "city_district": "manhattan",
  "city": "new york",
  "state": "ny",
  "country": "usa"
}
```

The geocoder would query Elasticsearch with the following combinations of fields identified by libpostal:

House Number	Street	Borough	City	State	Country	Layer
30	west 26th street	manhattan	new york	ny	usa	address
	west 26th street	manhattan	new york	ny	usa	street
		manhattan	new york	ny	usa	borough
			new york	ny	usa	city
				ny	usa	region
					usa	country

This all happens in one Elasticsearch query to reduce the number of roundtrips between the API and Elasticsearch. Since all the parsed fields match (including house number and street), the result that matched everything is returned and everything else is thrown away.

Fallback to Street

Our engine currently only supports address points and not house number interpolation (under **active development** (http://pelias.io/quarterly_goals/q4-2016.html)), so when a house number isn't an address point in our data, our logic will fallback to trying to find the street instead. Much like a street that doesn't exist in a city, 32 W 26th Street, New York, NY isn't an address point in our data but W 26th Street, New York, NY does exist as just a street so it will return a street result instead.

Fallback to City

Another likely scenario happens when an input contains a street, city, and state but the street is either misspelled or doesn't exist in the city, for example, `Calle de Lago, New York, NY`. The street `Calle de Lago` has been identified by libpostal as a street but just does not exist in New York City. When the street lookup fails, `New York City` will be the fallback.

Fallback to State

Quick question (no cheating): What US state is Hilton Head Island in? If you said North Carolina, that's understandable, but it's actually in South Carolina. That's one of the most common city/state mismatches that users make. While we'll soon handle corrections like this, but until then the fallback logic is to match the state only. That is, the input `Hilton Head Island, North Carolina` will fall back to `North Carolina`.

You can see where this is going

Listing all the possible fallbacks and failure scenarios is too exhaustive for a blog post but the approach that Pelias takes with an input is to first try the most specific combination of analyzed fields, then fallback to less granular combinations until a result is returned.

Catastrophic Failure

If the engine is unable to come up with any results based on the libpostal interpretation, it will revert to the old behavior. For example, the input `10 Main Street, United States of America` is parsed as a street/country but our data only supports `United States` and `USA` (alternate names support will be added in the near future) so no results would be returned. In this case, the engine falls back to the old behavior of finding *some* tokens from `10 Main Street, Fair Haven, VT, USA` to `10 Main Street, Swanland, England, United Kingdom`.

Incremental Improvements

We're starting our transition incrementally. It's certainly possible to release an epic update all at once but we'd risk alienating users whose applications may not be ready for such a monumental change and, as an open source project, we value the input of our users.

For our first iteration, we'll only be modifying behavior for inputs that libpostal says contain a street. That is, the new rules will apply for inputs like:

- `30 West 26th Street, New York, NY`
- `West 26th Street, New York, NY`

- 30 West 26th Street

but not:

- New York
- pizza in Lancaster, PA

Inputs like the latter set are parsed correctly but involve introducing advanced business rules for ranking. By concentrating our efforts on the former, we see what works and what doesn't for our users. When an input does not contain a street then the geocoder will resort to the original behavior.

Accuracy flags

With the introduction of the new fallback behavior, it became important to signal the user when it occurred. To accomplish this, We've added two additional properties to results of all queries processed using libpostal: `accuracy` and `match_type`. These fields, in combination with the `confidence` score, give a sense of how relevant and accurate the results are.

The `accuracy` property indicates what type of geometry the result contains. When `accuracy` is set to `point`, the result is either a POI or address that was represented with a single point in the source data. When `accuracy` is set to `centroid`, however, the point representing the location of the record is a centroid of what was once a polygon in the source data. The `centroid` accuracy type is common across streets and all administrative areas, such as cities, counties, regions, etc.

The `match_type` property has been added to indicate how precisely we've managed to match the user's requested address. In the case where everything lines up just perfectly, the `match_type` property will be set to `exact`. However, if for some reason the housenumber requested isn't found the result will be a fallback to the centroid of the matching street, and the `match_type` will be set to `fallback`.

Example time! Let's first see what happens when everything goes smoothly.

30 West 26th St, New York, NY will result in

```
[...]  
  "confidence": 1,  
  "match_type": "exact",  
  "accuracy": "point",  
[...]
```

Now if we look for a housenumber that doesn't exist on that street, like 1 West 26th St, New York, NY , you will see that the result is the centroid of West 26th Street and has the following properties

```
[...]  
  "confidence": 0.8,  
  "match_type": "fallback",  
  "accuracy": "centroid",  
[...]
```

Well, what if we screwed up the street name also and it doesn't exist in New York, maybe like 30 West 2006th St, New York NY ? Search will have no choice but to fallback to city level and return the centroid of New York City with the following properties. Note that the confidence score is significantly lower in this case because the result is less granular than a street centroid.

```
[...]  
  "confidence": 0.6,  
  "match_type": "fallback",  
  "accuracy": "centroid",  
[...]
```

Keep in mind that the new `match_type` property will only be returned in cases where libpostal and the new query logic is employed. In the case where then engine still relies on textual matching the property will be omitted.

Get in touch!

Let us know what you think and what you see! Libpostal is constantly improving, but if you see results you don't expect, please **file an issue on Github** (<https://github.com/pelias/pelias>) or **chat with us on Gitter** (<https://gitter.im/pelias/pelias>).

· 12 October 2016 ·

**Stephen Hess**

Stephen works on geocoding exclusively as a means to fund his passion for designing and building wooden frames for old maps.

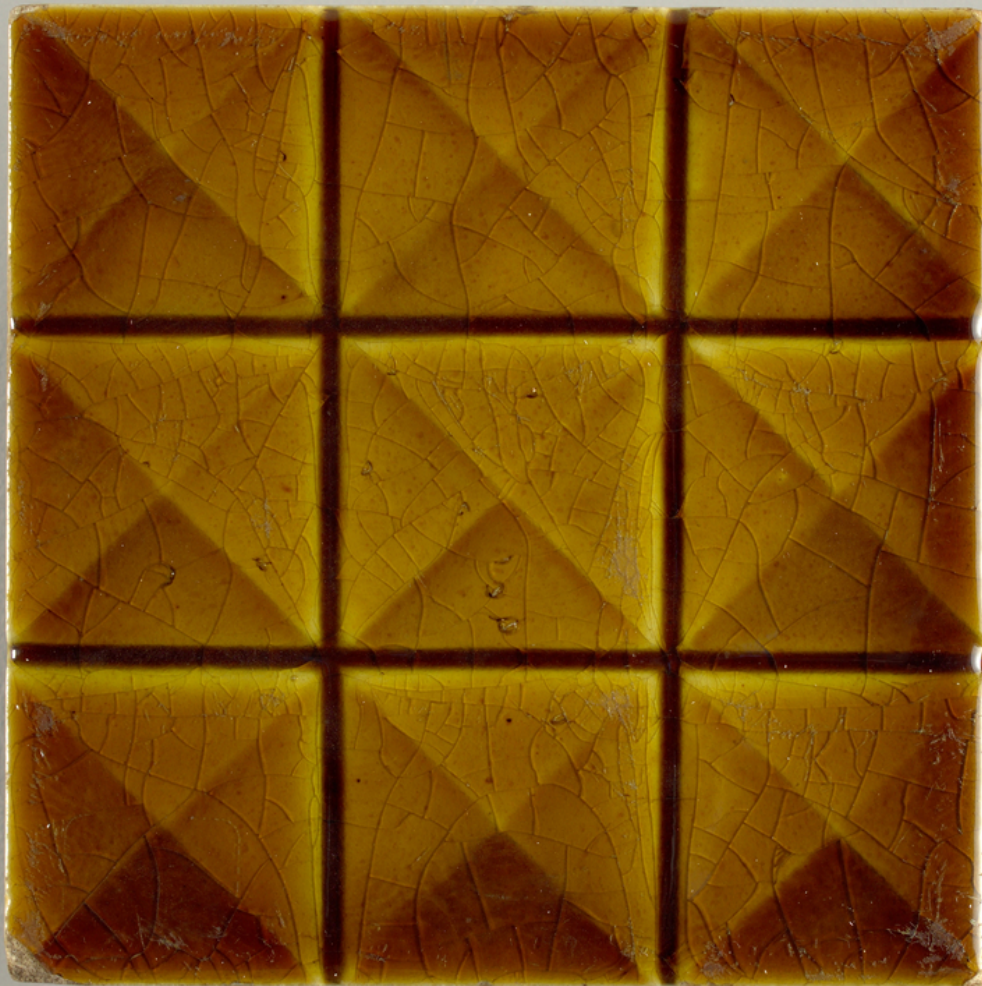
**Diana Shkolnikov**

Diana is the engineering director of Search@Mapzen and a proponent of mandatory fun, testing, education, and paleo, not necessarily in that order.

© 2017 Mapzen

Mapzen Vector Tiles at version 1.0

vector-tiles ([/tag/vector-tiles](#)) **documentation** ([/tag/documentation](#)) **data**
([/tag/data](#)) **tangram** ([/tag/tangram](#))



When we **launched** (<https://mapzen.com/blog/look-upon-our-squares-of-math-in-three-dimensions/>) vector tiles back in May 2015 it was essentially a TV pilot episode. The plot, characters and direction were compelling enough to get signed for a full season and we took it to production. Today we're proud to show you v1.0 of Mapzen's Vector Tile service!

After 20 months in active development, v1.0 vector tiles is an important milestone and introduces stability to the service API and the underlying Tilezen data model. To get there we had to break a few things. The full story is in the v1.0.0 **changelog** (<https://github.com/tilezen/vector-datasource/blob/master/CHANGELOG.md>), summarized below.

What's evolved since the pilot? Our cast of characters is now at 400 major roles and over 200 minor roles (up from just 200 major roles in v0.1). Tolstoy would be proud! And yes, we have names in Russian, **Kannada** (<http://thejeshgn.com/2016/09/24/kannada-web-maps-using-tangram-by-mapzen/>), and all languages.

Data is organized into 9 layers:

- boundaries , buildings , earth , landuse , places , pois , roads , transit , and water

Like chapters in a book, we reveal data through 18 different map zooms. Zoom 0 fits the world onto your smartphone screen while panning into zoom 18 reveals neighbourhood streets, buildings, and businesses. We've spent an insane amount of time orchestrating when each of those 600+ features come in and out of view, how prominent each is, and we've sourced quality data for each role.

We've obsessed over the details: label boundaries with left and right names, realistic 3D buildings with heights and building parts, points of interest selected for display at low- and mid-zooms to support wayfinding, layering of complex highway interchanges based on overpass and underpass tags, support for advanced road shields, bicycle maps, hiking maps, landuse kind intercuts for roads and buildings, transit line colors, bus routes, transit stations, intermittent streams, coastline boundaries, maritime boundaries, and more.

The focus of v1.0 vector tiles has been four fold:

1. New URL scheme!
2. Smaller file sizes, especially at low- and mid-zooms for better client-side performance.
3. Data model changes for improved consistency of `kind` values and other properties across all zoom levels, and introduce a new `kind_detail` property that aggregates

across multiple OpenStreetMap source tags for layers like roads.

4. Bug fixes from the v0.10 release focused on outdoor mapping, adding new features such as camp sites and rest areas, and more valid polygon geometries (especially for MVT).

New URL scheme

Mapzen now offers several different types of tiles in vector and raster formats and we combine data from multiple sources. The API endpoints have been updated to reflect this, and emphasize versions and account names.

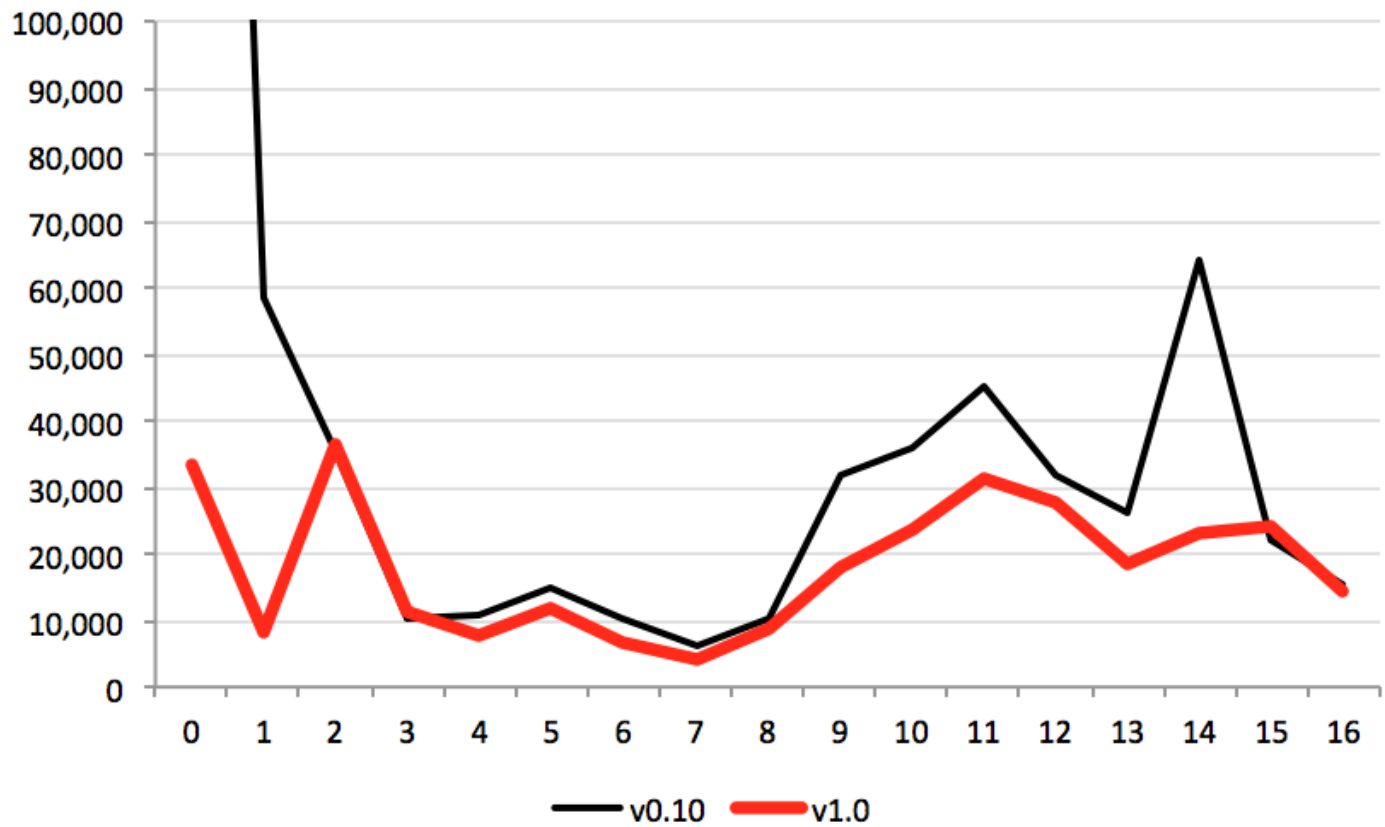
- **GeoJSON:** `http://tile.mapzen.com/mapzen/vector/v1/all/{z}/{x}/{y}.json?api_key=mapzen-xxxxxxx`
- **TopoJSON:** `http://tile.mapzen.com/mapzen/vector/v1/all/{z}/{x}/{y}.topojson?api_key=mapzen-xxxxxxx`
- **Mapbox Vector Tile:** `http://tile.mapzen.com/mapzen/vector/v1/all/{z}/{x}/{y}.mvt?api_key=mapzen-xxxxxxx`

Legacy tile support

Your existing maps should continue to work, but we recommend upgrading sooner than later. The old production URLs will remain available for the next year but will no longer show the most up-to-date OpenStreetMap data, render new coverage areas on the fly, or return custom layers (only `all` and `buildings` layers are archived). Upgrading your maps to v1.0 is straightforward though: **here's the cheatsheet we used internally** (<https://gist.github.com/nvkelso/a7cfec649bb0de49f4ac71596a148a32>) – let us know if you have any questions!

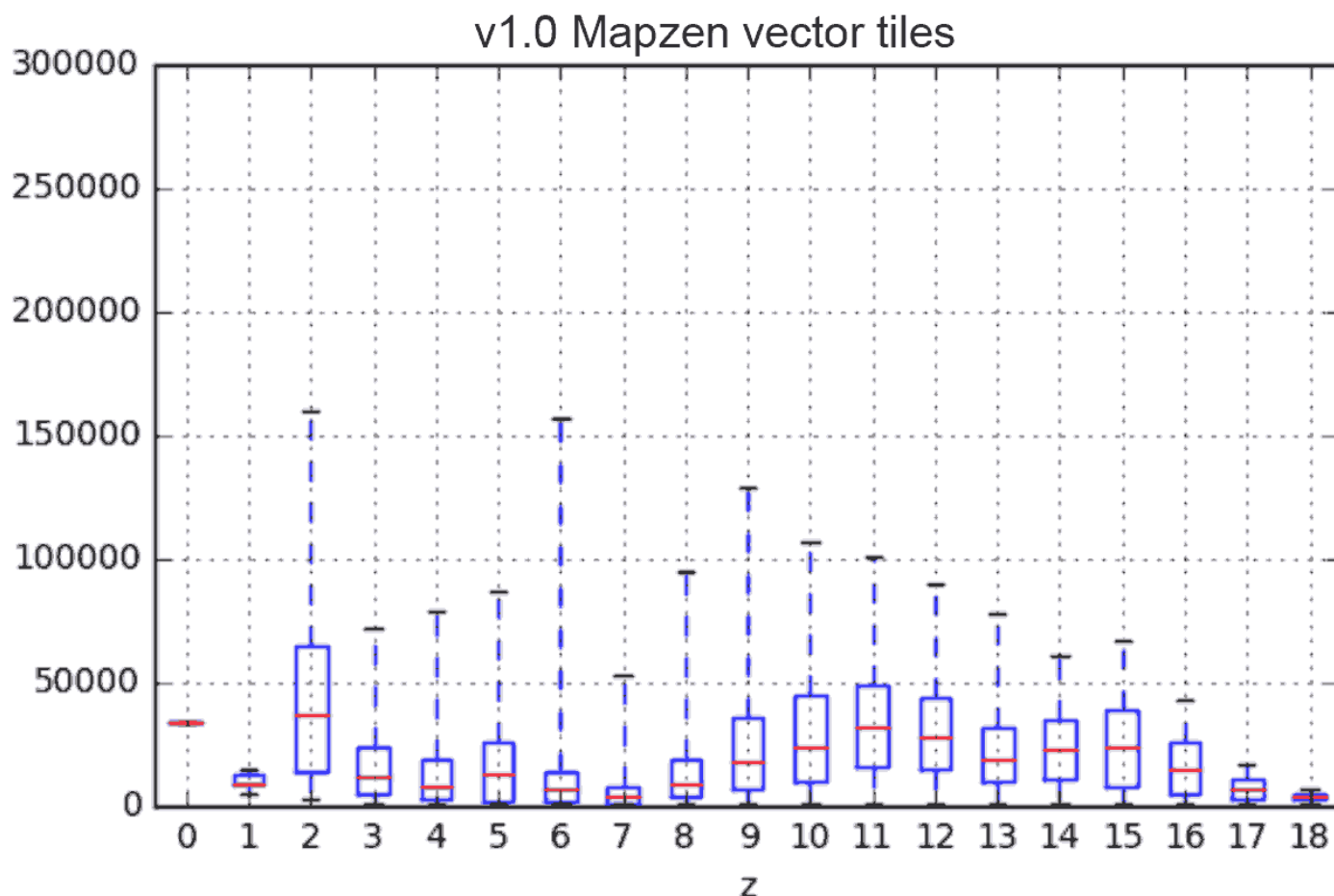
Smaller file sizes

By removing unnecessary features from low-zooms and merging landuse and buildings at mid- and high-zooms we've reduced average tile size by 30% and in some cases by 90%. As can be seen in the graph below, our median tile sizes are now under 40kB across all zooms.



Average tile file size by zoom, in kilobytes (256px tiles)

When measuring the 99th% worst case, we see even better improvements to file size. Most large tiles are now more than 50% smaller than version v0.10 tiles.



Variability in tile size by zoom – red line shows median, box out to the 1st and 3rd quartiles, and whiskers out to the 1% and 99% percentiles, in kilobytes (256px tiles)

We monitor these stats using the open source **Scoville** (<https://github.com/tilezen/scoville>) collection framework.

Data model consistency

The Mapzen Vector Tiles service is powered by **Tilezen** (<https://github.com/tilezen/>), our open source tiles project.

Version v1.0 has focused on seamlessly blending data (essentially the `kind` values) across the different sources per zoom, ensuring a consistent data model regardless of data source. We also introduce a new `kind_detail` property that aggregates across multiple OpenStreetMap source tags for layers like roads.

Mapzen vector tiles mash together several major open data sources and we owe a tremendous debt of gratitude to the individuals and communities which produced them. Specifically **Natural Earth** at low-zooms, **OpenStreetMap** at mid- and high-zooms, and **Who's On First** at high-zooms.

Each data source has its own data model, and earlier versions of the vector tile service took a hands off approach, passing through raw or lightly transformed values from each. With version v1.0 of the Mapzen vector tile service we now normalize all kind values into a standard Tilezen data model based primarily on OpenStreetMap convention.

Attribution is required for many of our data sources. See the **Attribution** (<https://mapzen.com/documentation/vector-tiles/attribution/>) documentation for more information.

Data model interoperability

While the Tilezen data model used in Mapzen vector tiles is optimized for display, we have taken a generic interoperability approach to support many different map styles and map renderers besides our own **Tangram** (<https://mapzen.com/products/tangram/>). Please copy our data model!

The Tilezen **semantic versioning** (<https://github.com/tilezen/vector-datasource/blob/master/SEMANTIC-VERSIONING.md>) documentation details the existing data properties, values, and our promises around them and is excerpted below.

Tilezen data model overview:

- **common** - layers, property names, and kind values are generally available across all features in a Tilezen response.
 - Establishes basic selection of features and their arrangement into specific named layers.
 - Core properties needed for display and labeling of features:
 - Special bits that make vector tile content **interoperably Tilezen**, including `kind` , `kind_detail` , `landuse_kind` , `kind_tile_rank` , `min_zoom` , `max_zoom` , `is_landuse_aoi` , `sort_rank` , `boundary` , and `maritime_boundary` .
 - Fundamental properties like `name` (including localized names), `id` , and `source` included on most every feature.
- **common-optional** - These are meant to be part of a common set, but may not be present because they aren't relevant or because we don't have the data (primarily feature properties).
 - Used to refine feature selection.
 - Lightly transformed **interoperable Tilezen** properties based on original data values. Examples include: `country_capital` , `region_capital` , `bicycle_network` , `is_bridge` , `is_link` , `is_tunnel` , `is_bicycle_related` , `is_bus_route` ,

walking_network , area , left & right names and localized name:* values on lines, and left & right id values on lines.

- Fundamental properties like ref , colour , population , elevation , cuisine , operator , protect_class , and sport .
- **optional** - These are the properties of a specific, less important kind of feature, or generally present across all features of a kind but only in exceptional cases.
 - Often used to decorate features already selected for display.
 - Additional properties like capacity and covered .

Get an API key

To use the Mapzen Vector Tiles service, you should first get a developer API key. Sign in at **<https://mapzen.com/developers>** (**<https://mapzen.com/developers>**) to create and manage your API keys, it'll only take a minute (or two).

1. Go to **<https://mapzen.com/developers>** (**<https://mapzen.com/developers>**).
2. Sign in with your GitHub account. If you have not done this before, you need to agree to the terms first.
3. Create a new key, and optionally, give it a name so you can remember the purpose of the project.
4. Copy the key into the terrain URL where mapzen-xxxxxxx represents your key.

Tell me more

Complete documentation for the Mapzen Vector Tiles service is available from:

<https://mapzen.com/documentation/vector-tiles/>
(<https://mapzen.com/documentation/vector-tiles/>)

We also have a Gitter chat room: **join the conversation** (**<https://gitter.im/tilezen/tilezen-chat>**).

Curious how Tilezen works or want to help out? The **contributing** (**<https://github.com/tilezen/vector-datasource/blob/master/CONTRIBUTING.md>**) docs are worth a read.

General improvements

We've improved a few things since v0.10:

- **tilejson**: Major upgrade to reflect all layers and properties for applications like Mapbox Studio and Tangram Play.
- **general**: More geometries are valid, especially in MVT format.
- **general**: Better attributing of `source` to OpenStreetMap for pois, transit, and other layers.
- **roads** layer: To support highway shields a new `shield_text` property has been added and existing `network` values have been normalized. Multiple shields are supported via optional `all_networks` and `all_shield_texts` lists. (The `ref` property remains available but is less useful for shield construction.)
- **boundaries** layer: New `map_unit` lines.
- **buildings** layer: Features of `building_part` may receive a `root_id` corresponding to the building feature, if any, with which they intersect.
- **pois** and **landuse** layers: Added `graveyard` , `camp_site` , and `art_gallery` .

Breaking changes

Going forward we will adhere to semantic versioning (semver) and have added **documentation** (<https://github.com/tilezen/vector-datasource/blob/master/SEMANTIC-VERSIONING.md>) detailing the promises the open source Tilezen project makes about major, minor, and patch versions and data model changes.

Of special note in the v1.0 release, the draw order of map features should now be keyed off `sort_rank` (was `sort_key`), language codes are normalized to 2-chars ISO languages, and city place labels (now `locality`) and state boundaries lines (now `region`) will break unless care is taken to update map stylesheets.

Pay special attention to low-zoom `kind` values – they were originally passed through raw from Natural Earth but are now normalized. We also dropped `scalerank` values from Natural Earth, mapping them instead into `min_zoom` values and added `min_zoom` values for all features in all layers.

Additionally, developers who used to parse buildings tiles for urban modeling at zoom 13 will no longer get back full data (use zoom 16 instead).

Getting to final v1.0 tiles we've broken a few things:

- **boundaries** layer: New `locality` kind is used for all city-like features, and `region` is used for state- and province-like features.
- **buildings** layer: features are merged at zooms 13, 14, and 15, and `height` values are quantized and some properties are removed at those zooms (but a

new `scale_rank` property is added to distinguish groups of big buildings from groups of small buildings). Values previously found in `kind` are moved to `kind_detail`, and a value whitelist is applied.

- **landuse** layer: Rename several `kind` values to distinguish `natural_wood` from `wood`, `natural_forest` from `forest`, and `natural_park` from `park`.
- **landuse-labels** layer: has been removed in favor of features in the **landuse** layer and **pois** layers. (We marked it as deprecated several releases ago.)
- **places** layer: To reduce file sizes country labels were removed from zooms 0 and 1.
- **places** layer: New `locality` `kind` is used for all city-like features, and `region` is used for state- and province-like features. Capitals are now indicated as `country_capital` (was `capital`) and `region_capital` (was `state_capital`).
- **pois** layer: Some kinds were renamed to distinguish `aeroway_gate` from `gate`, specify `gas_canister` shops (was `gas` which was confusing with automotive gas stations), and split off `ski_rental` if a `ski` feature was primarily a rental facility.
- **transit** layer: `route_name` on line geometries is now simply `name`.
- **Natural Earth kind values:** Features sourced from Natural Earth now have normalized `kind` value in layers like boundaries, landuse, places, and roads layers.
- **OpenStreetMap kind values:** All `kind` values in the **places** and **boundaries** layers now follow Who's On First placetype hierarchy.
- **roads** layer: Remove `aerialway`, `highway`, `piste_type`, and `railway` in favor of coalescing their values into a new `kind_detail` property.
- **pois** and **landuse** layers: `cuisine` and `sport` properties are dropped but their values moved to new `kind_detail` property.
- **pois** and **landuse** layers: Reclassify some national `forest`, `park`, `national_park` and other "green" areas based on their `operator`, `protect_class`, and other factors (and normalize `operator` values).
- **pois** and **landuse** layers: Change `min_zoom` so smaller, less important features are only included at later map zooms.
- **places** layer: Adjust `min_zoom` values for Natural Earth at the low-zooms and Natural Earth and OpenStreetMap `localities` at mid- and high-zooms to reduce label crowding.
- **buildings** and **landuse** layers: remove label placements from low- and mid zooms. Important landuse labels moved to the pois layer; the remainder are intended for text only labeling.

Read the docs

Check out the full story in the **v1.0.0 changelog** (<https://github.com/tilezen/vector-datasource/blob/master/CHANGELOG.md>) for new features and even more breaking changes.

Like any good drama, the Mapzen Vector Tile service has evolved over the season, with more than a dozen plot twists since v0.1 that we've listed below. Here's to v1.0 and the upcoming season!

Vector tile timeline

- **v0.1** (May 2015): Initial release, largely based on Michal Migurski's vector tile project [**1** (<http://mike.teczno.com/notes/postgreslessness-mapnik-veciles.html>) and **2** (<http://mike.teczno.com/notes/vector-tile-rendering-numbers.html>)]. **blog post** (<https://mapzen.com/blog/look-upon-our-squares-of-math-in-three-dimensions/>)
- **v0.2pre** (July, Aug 2015): Added dozens of new features in the **landuse** and **pois** layers and add extra tags like `sport`, `religion`, and `cuisine`. Also refined high-zoom feature visibility.
- **v0.2** (Sept 2015): Add `landuse_kind` intercut for roads. Add water boundary (coastline) lines. Add address labels.
- **v0.3** (Sept 2015): Show early national park and park polygons. Refine transition from Natural Earth to OpenStreetMap. Add `sort_key` for landuse polygons to prevent z-fighting.
- **v0.4** (Oct 2015): adjust balance of pois and landuse labels, add de-duplicated label positions for water and buildings. Boundaries based on OSM relations include left and right localized names, and maritime boundaries. Added ferry and airport runways lines.
- **v0.5** (Nov 2015): Switched to using neighbourhoods from Who's On First, started refining High Road logic for more rail, aerialway, pedestrian features. Stopped duplicating buildings into the landuse layer. Filter out duplicate feature labels in pois, landuse, and buildings layers. Add `kind_tile_rank` to stations.
- **v0.6** (Dec 2015): Hospitals, schools, and parking lot fixes. Added beaches, highway exits, road junctions, ice creme shops, toy stores, wine shops, alcohol shops, railway platforms, and pier lines. Refine zoom ranges for subway stations and zoos. Add winter sports pistes. Adds IATA codes to airports.
- **v0.7** (Jan 2016): Road merging at low- and mid-zooms. Adds additional POIs for fitness centers, swimming pools, prisons, and airport gates. Clip buildings to 3x tile. Zoo and tourist attraction fixed up.
- **Jan 2016**: Station relations **blog post** (<https://mapzen.com/blog/station-relations/>)
- **Feb 2016**: Blog post recapping v0.7, preview v0.8, and promoting new documentation.** **blog post** (<https://mapzen.com/blog/vector-tiles-v0-8-preview/>)
- **v0.8** (March 2016): Move to zoom 16 max tile (was 18). Add and revise `sort_key` values on all layers to reduce z-fighting. Add pois for International Women's Day including childcare, dentist, doctor, toilets and slew of other features including more water labels, hardware stores. Revise station `tile_kind_rank` with more data.

- **Mar 2016:** Optimized global content distribution network for Europe, USA, and Asia. Dropped OpenScienceMap format.
- **v0.9** (March 2016): OSM Transit! More revisions to station tile_kind_rank based on transit types and adjust transit feature zoom ranges and optional root_relation_id. Indicate if road is used for bus routes. Big SQL > CSV filter logic refactor.
- **v0.10** (May 2016): Huge number of hiking, pedestrian, and bike improvements for Walkabout style, including early zoom paths. Waterfall, peaks islands. Start normalizing all Natural Earth and OpenStreetMap kinds and sources. More CSV > YAML filter logic refactor, and database triggers. **blog post** (<https://mapzen.com/blog/v0.10-tiles/>)
- **June 2016:** Scoville starts monitoring production
- **June 2016:** Tilezen github org created!
- **June 2016:** Started enforcing vector tile keys (for on-demand path)
- **v1.0-pre1** (July 2016): Start of major breaking changes for v1.0 release. New URL endpoint, reclassify bunch of features roads, boundaries, places, pois, landuse, and buildings layers. Reduce file size by removing some features and by merging buildings at mid-zooms. Adds indoor corridors, sidewalks, toll booths, camp sites, rest areas, and localized neighbourhood names. Add ability to generated buffered MVT format.
- **v1.0-pre2** (Aug 2016): Low- and mid-zoom landuse merging. More normalization between Natural Earth and OpenStreetMap. Major changes for landuse and pois layers to improve consistency. Major changes to the places layer. Add new kind_detail property coalesce for roads features & etc. Adds explicate road shields. Fractional min_zooms values on all features in all layers. Major TileJSON upgrades. Semantic versioning doc.
- **v1.0-pre3** (Sept 2016): New URL pattern, again. Fix boundary min_zooms. Add min_zoom to all features. Major fixes for national park, parks, and forest kind assignments based on operator and protect_class. Major adjustments to zoom ranges for landuse and pois.
- **v1.0 final** (October 2016): Fix MVT format geometry errors in land & water.

Feedback

Have any feedback on the vector tiles? Please direct feedback to **Nathaniel Vaughn Kelso** (<https://github.com/nvkelso>), our Chief Cartographer, at **hello@mapzen.com** (<mailto:hello@mapzen.com>)!

*image via **Cooper Hewitt** (<https://collection.cooperhewitt.org/objects/18797939/>), Tile (USA), 19th–20th century; Manufactured by Trent Tile Company ; glazed stoneware; H x W x D (laying flat): 10.5 x 10.5 x 1.2 cm (4 1/8 x 4 1/8 x 1/2 in.); Gift of Florence Barnes; 1981-56-77*

**Nathaniel Vaughn Kelso**

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

**Matt Amos**

OpenStreetMap developer-contributor and chilli enthusiast. Writes vector tile and other data-mastication tools at Mapzen.

**Rob Marianski**

Software engineer working on cutting tiles and infrastructure. Functional programming enthusiast.

**Olga Kavvada**

Former Mapzen data team intern. Passionate about the outdoors and all things spatial.

**Ian Dees**

Ian is on the tiles team, building squares of map data. Ian ❤️ maps and bleeds nodes, ways, and relations.

© 2017 Mapzen

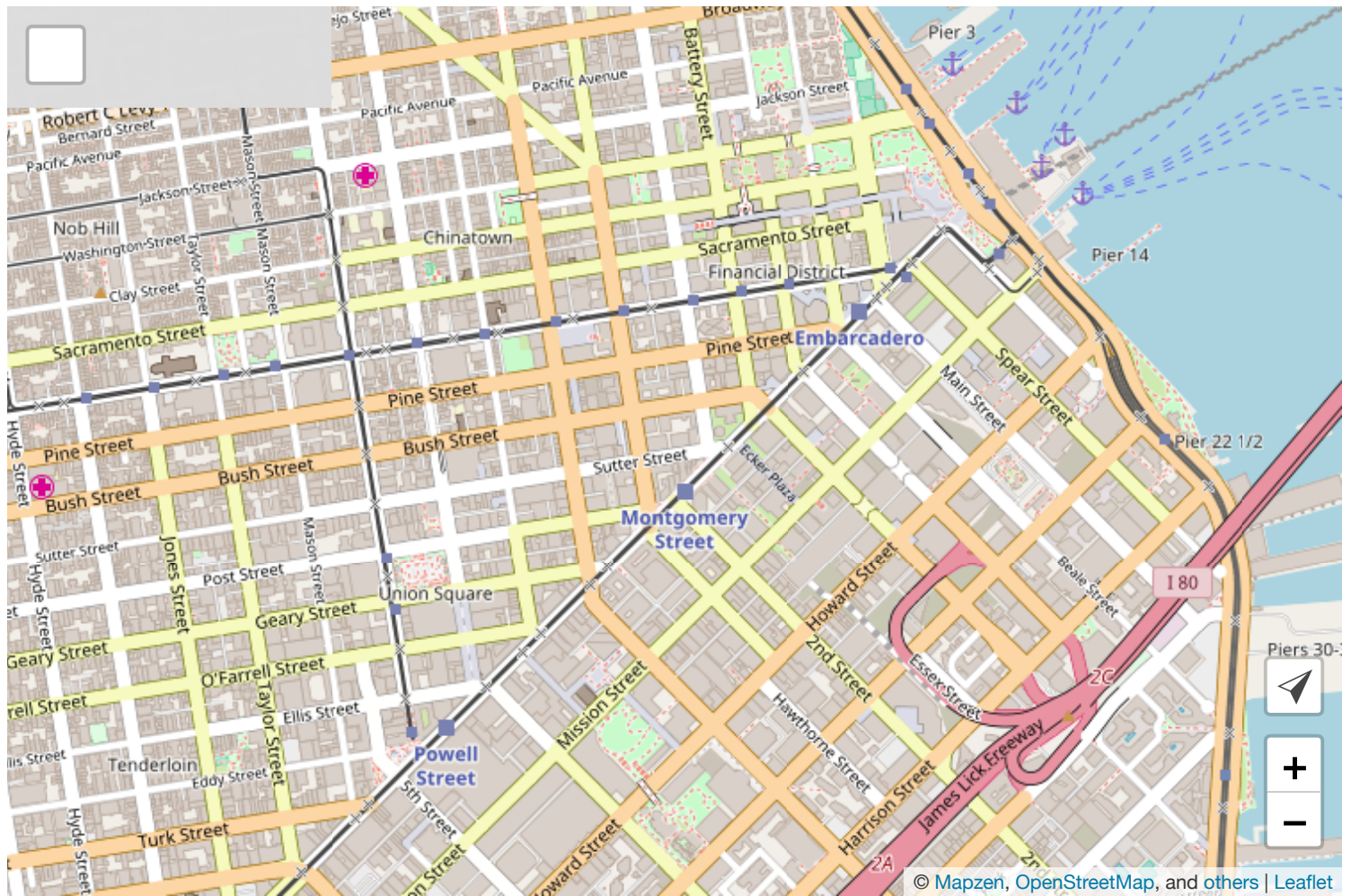
Tron 2.0 – Creating a Visual Language of Scale

tangram (/tag/tangram) vector-tiles (/tag/vector-tiles) cartography (/tag/cartography)

Today we introduce **TRON** version 2 as a fully realized Mapzen **house style** (<https://mapzen.com/products/maps/>), rebuilt from the ground up to take advantage of the latest features of the **Tangram** (<https://mapzen.com/products/tangram/>) graphics engine and Tangram **blocks** (<http://tangrams.github.io/blocks/>). With this new version, visual forms and elements transform per zoom, revealing new cartographic details and a deep exploration of scale transformations.

We're pushing both the medium and mapping to new extremes with TRON, and it *will* push your GPU and fan to the limit. We want to let everyone know what is possible in OpenGL and WebGL, and designed it for game developers to remix.

Will autonomous cars dream as they charge overnight? Mapzen's Tangram lets us imagine! Explore **TRON** v2 in this interactive map, and keep reading to dive into its *visual language of scale*.



(Open San Francisco (<https://mapzen.com/products/maps/tron/more-labels/#lat=37.7926&lng=-122.4003&z=15>) full screen. Or Tokyo (<https://mapzen.com/products/maps/tron/more-labels/#z=15.270148228690053&lat=35.6848&lng=139.7536>)! Or Paris (<https://mapzen.com/products/maps/tron/more-labels/#z=15.487369566329281&lat=48.8549&lng=2.3453>)!)

WHAT ARE THE VISUAL QUALITIES AND ELEMENTS OF THE TRON UNIVERSE?

GLOW

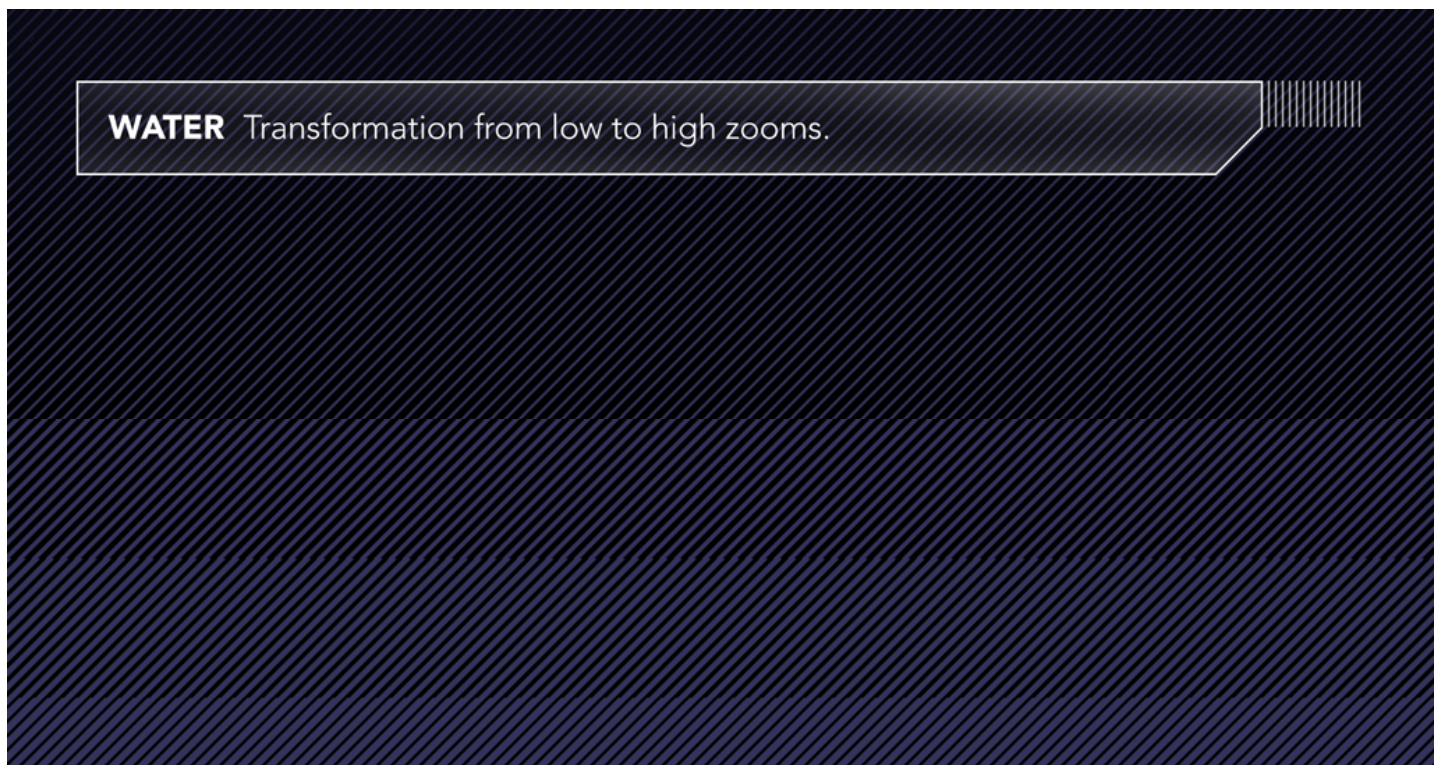
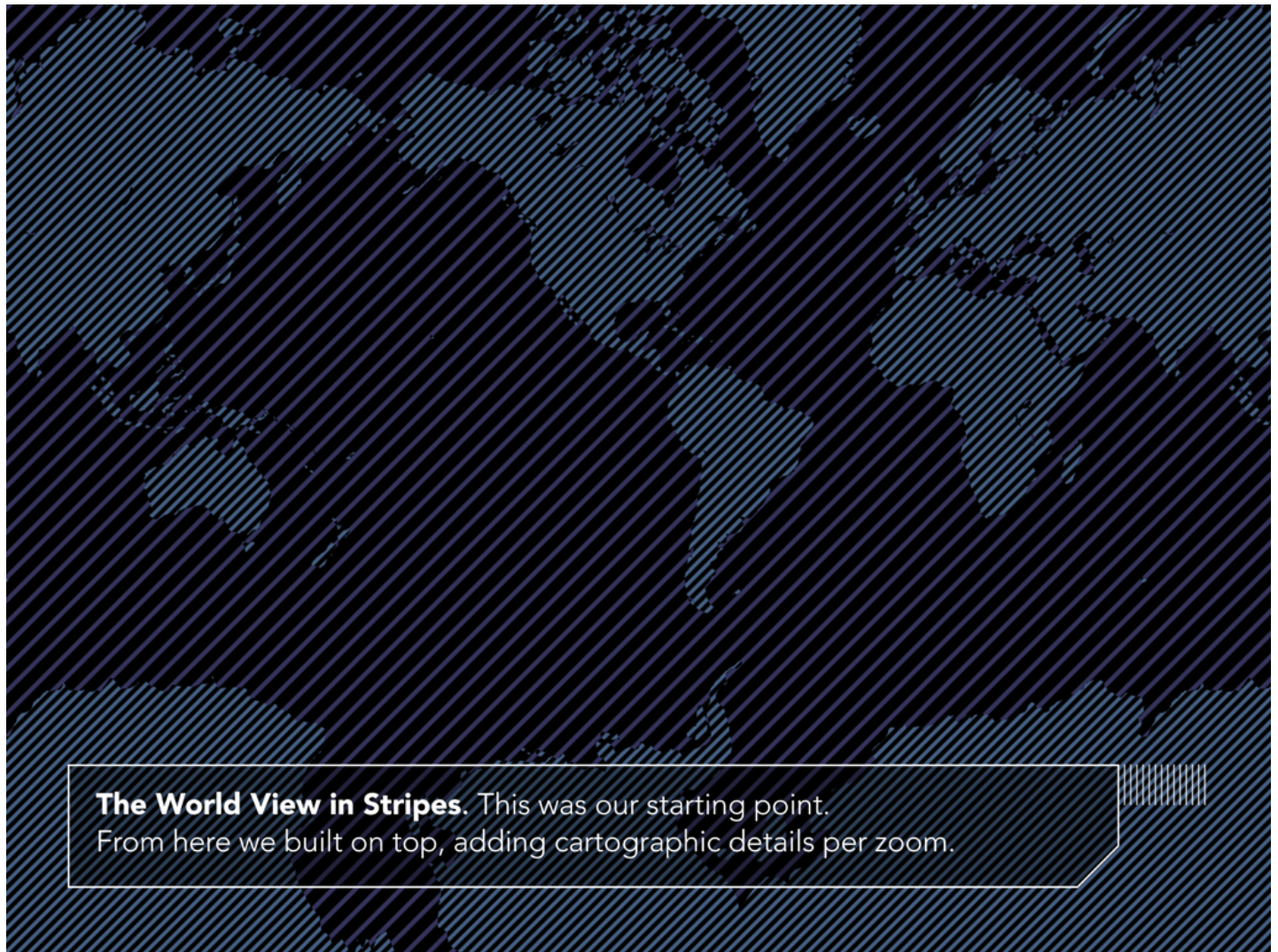


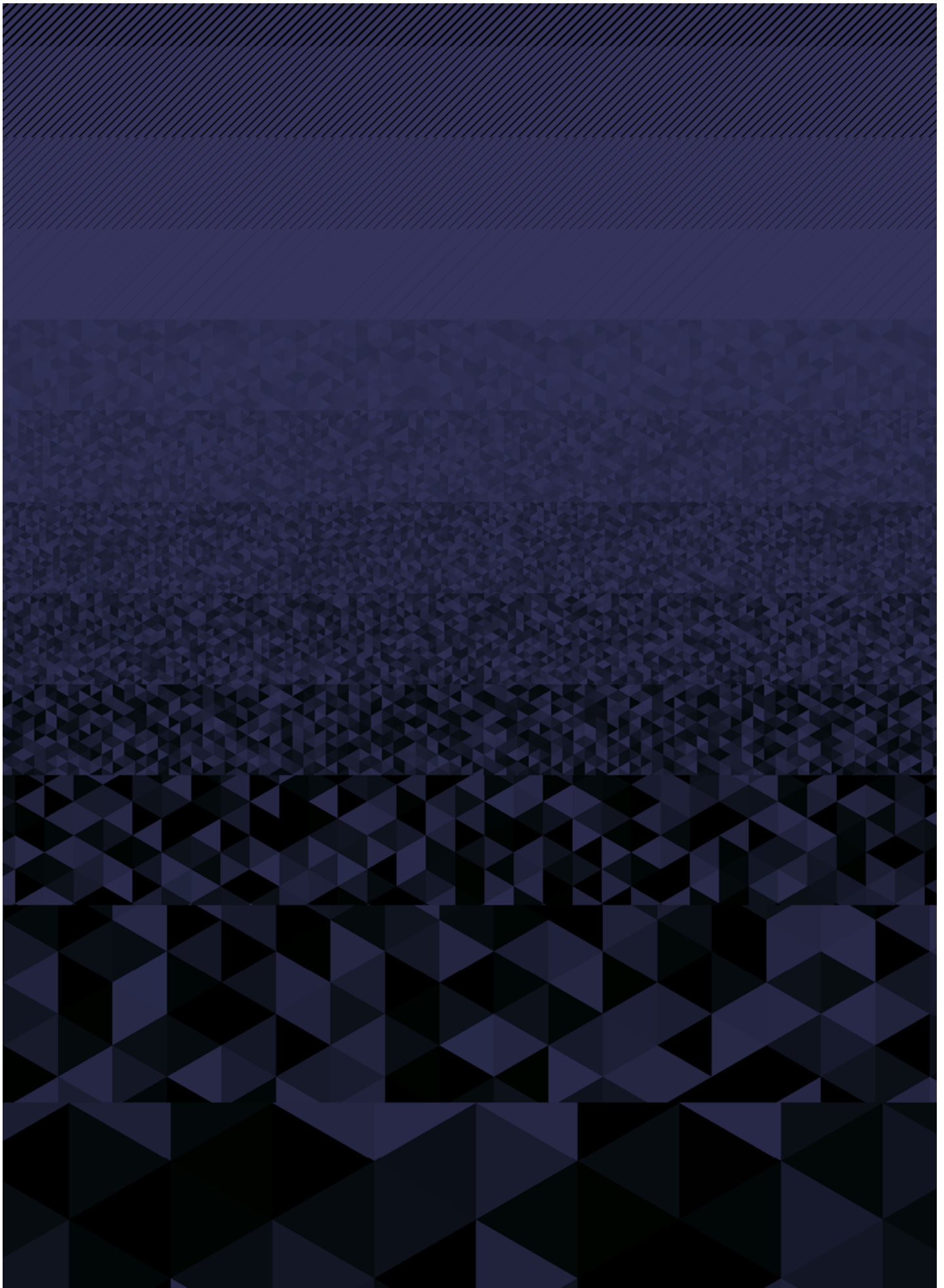
PRIMARY SHAPES



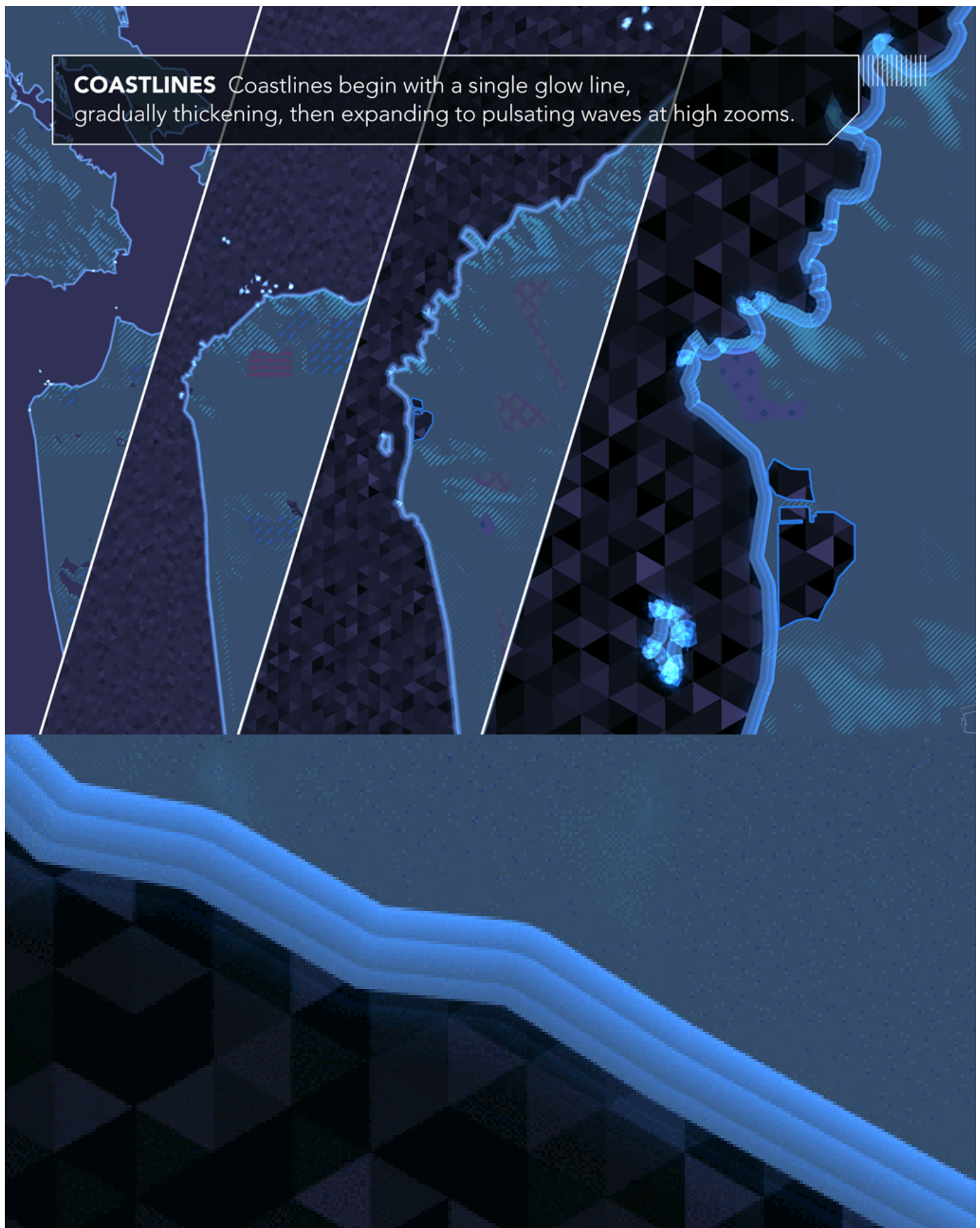
REPETITION

HOW CAN WE TRANSLATE THIS VISUAL VOCABULARY
TO EXPRESS THE EXPERIENCE OF SCALE?



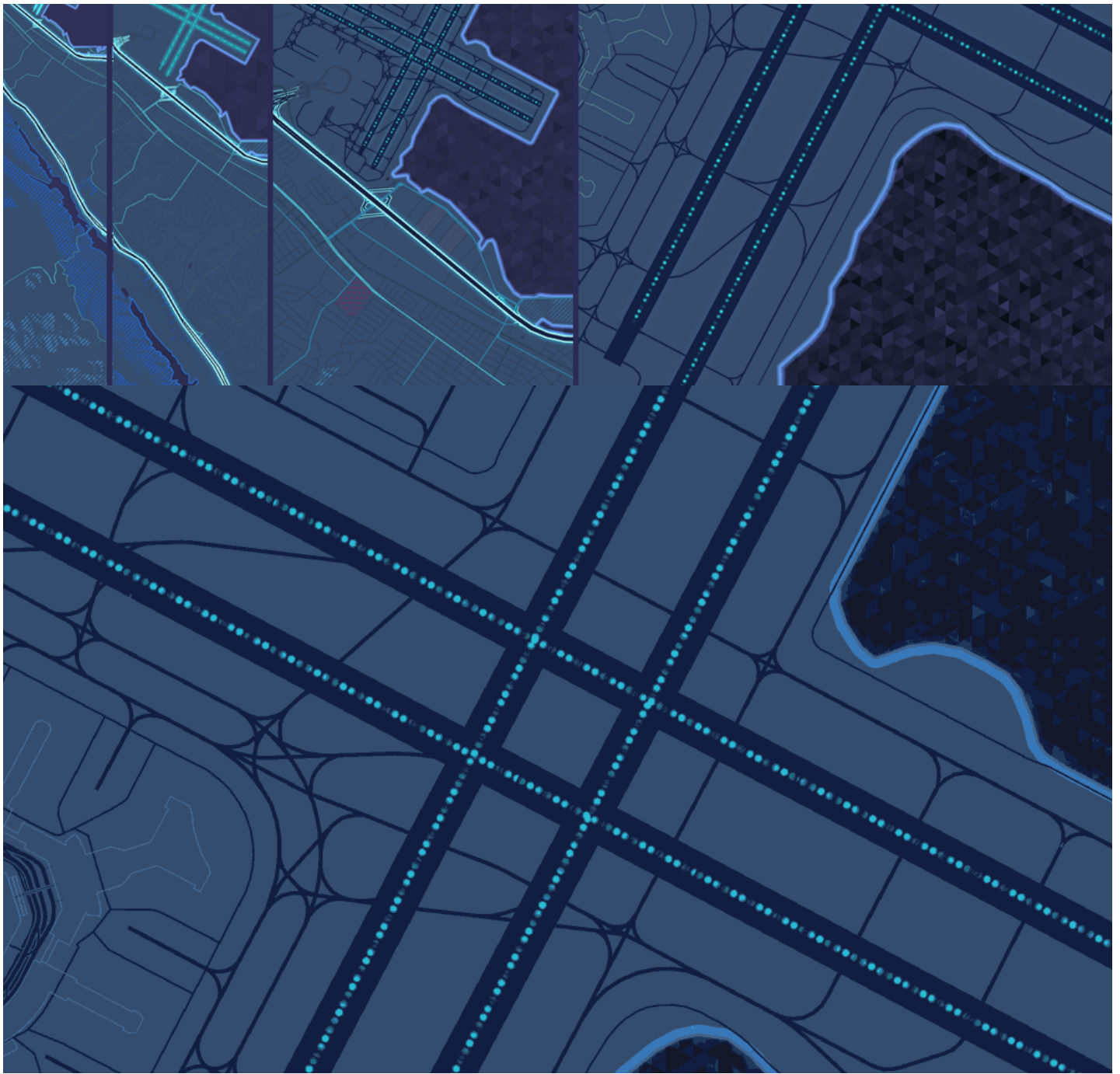




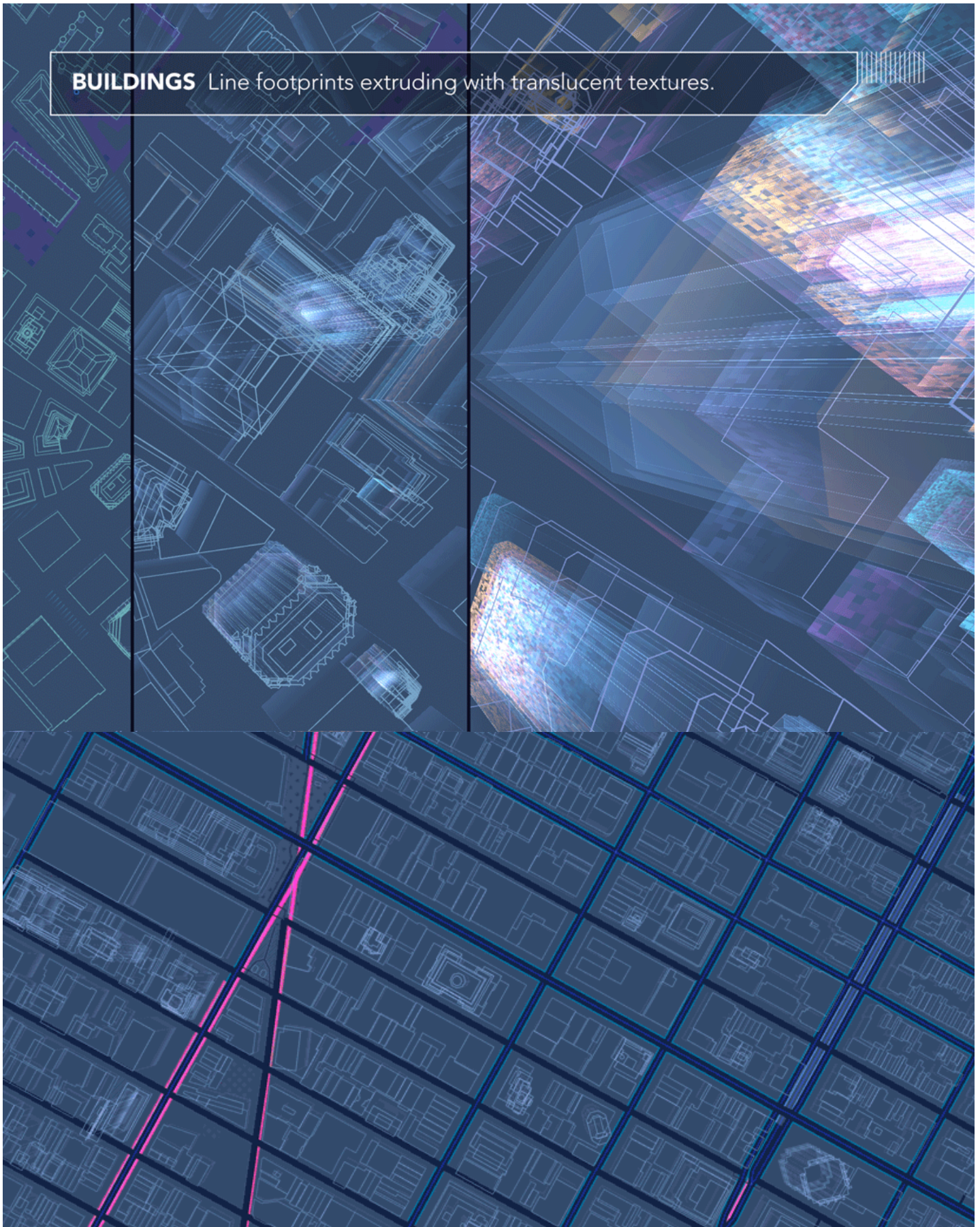


AIRPORTS Airports begin as single glow lines transitioning to thick dark lines, revealing runway lights at higher zooms.

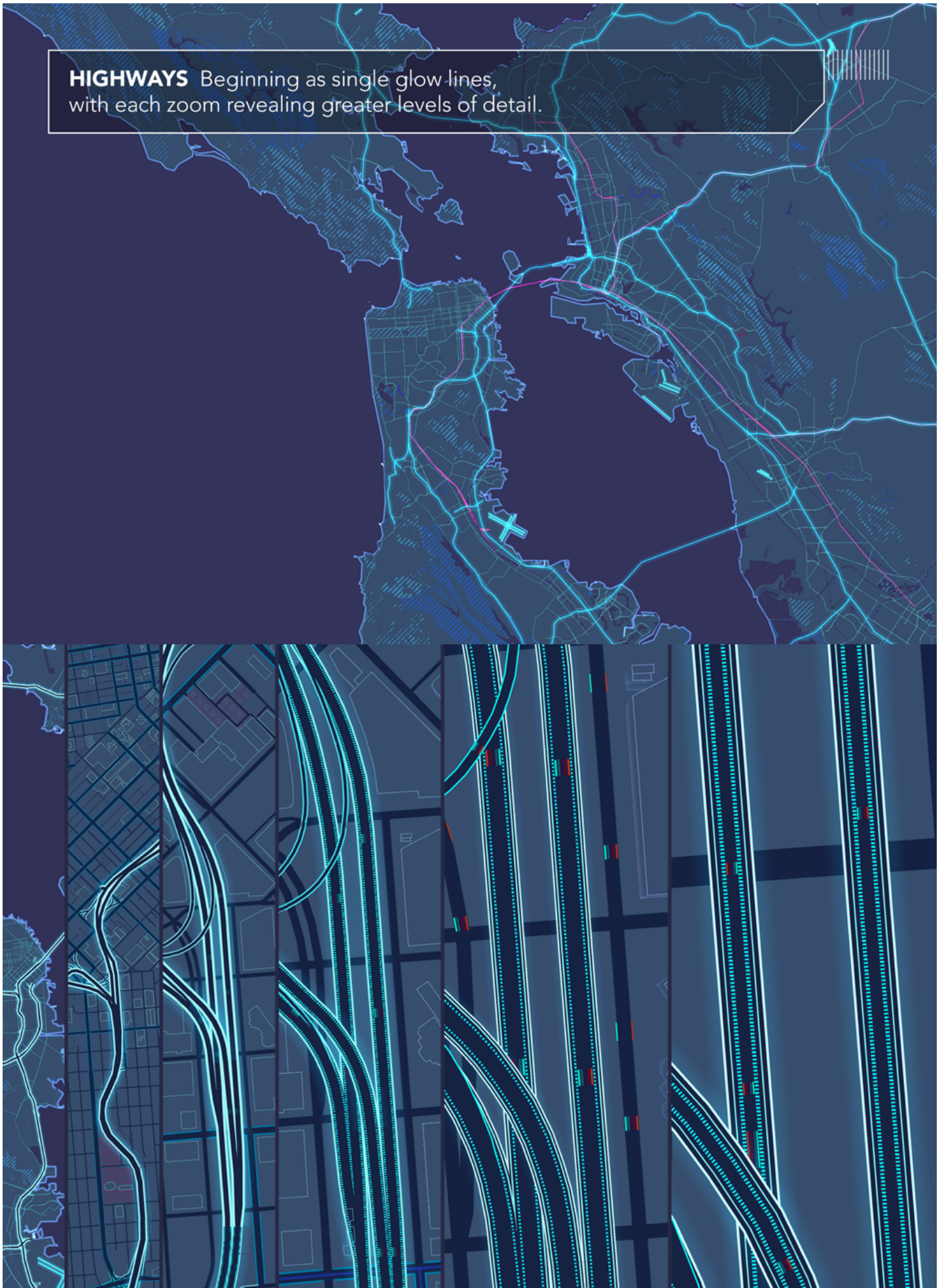




BUILDINGS Line footprints extruding with translucent textures.



HIGHWAYS Beginning as single glow lines, with each zoom revealing greater levels of detail.







· 13 October 2016 ·



Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.



Patricio Gonzalez Vivo

Patricio is an artist and graphic engineer who speaks the language of light.

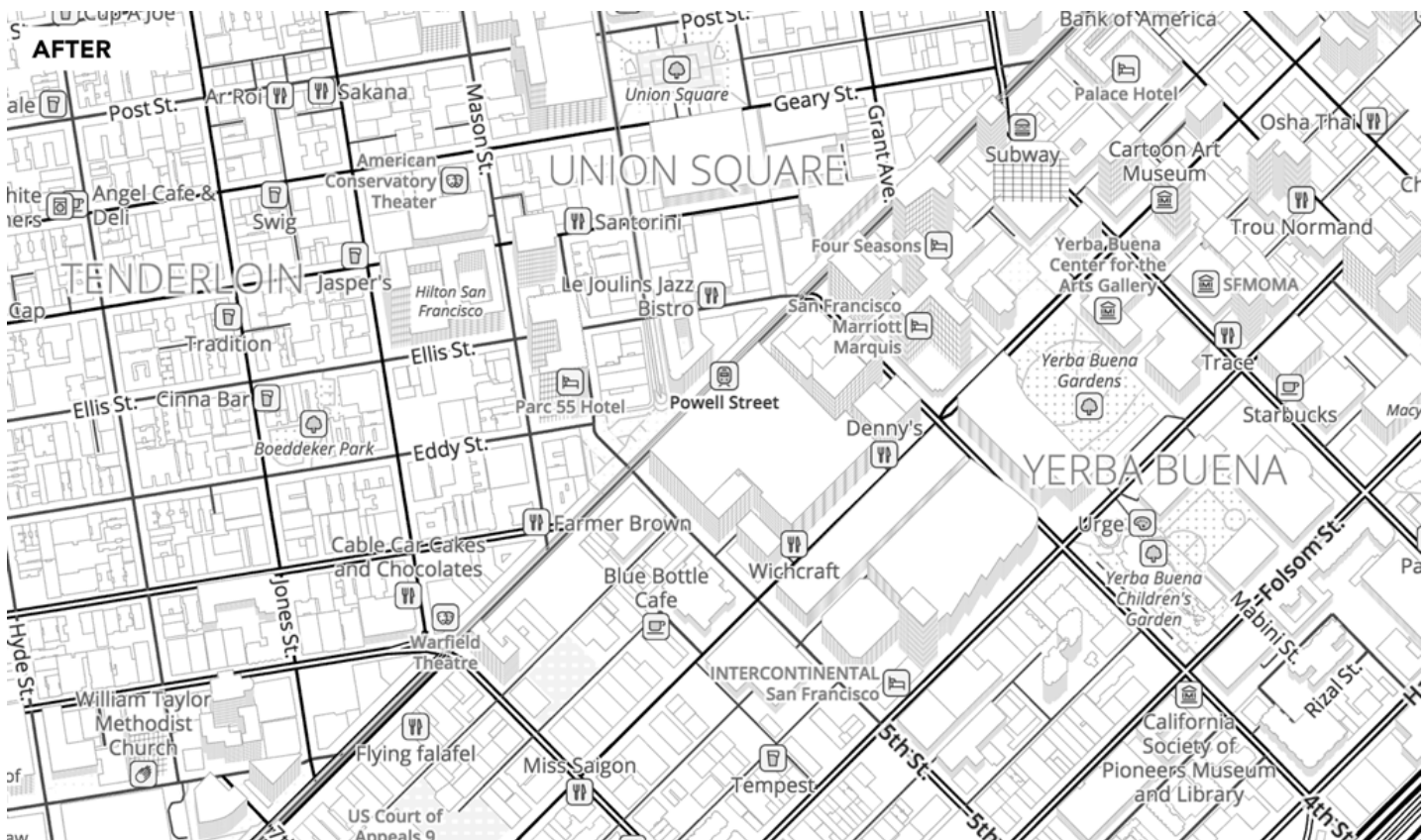
© 2017 Mapzen

Updated Mapzen House Styles – Better labels, new v1.0 vector data

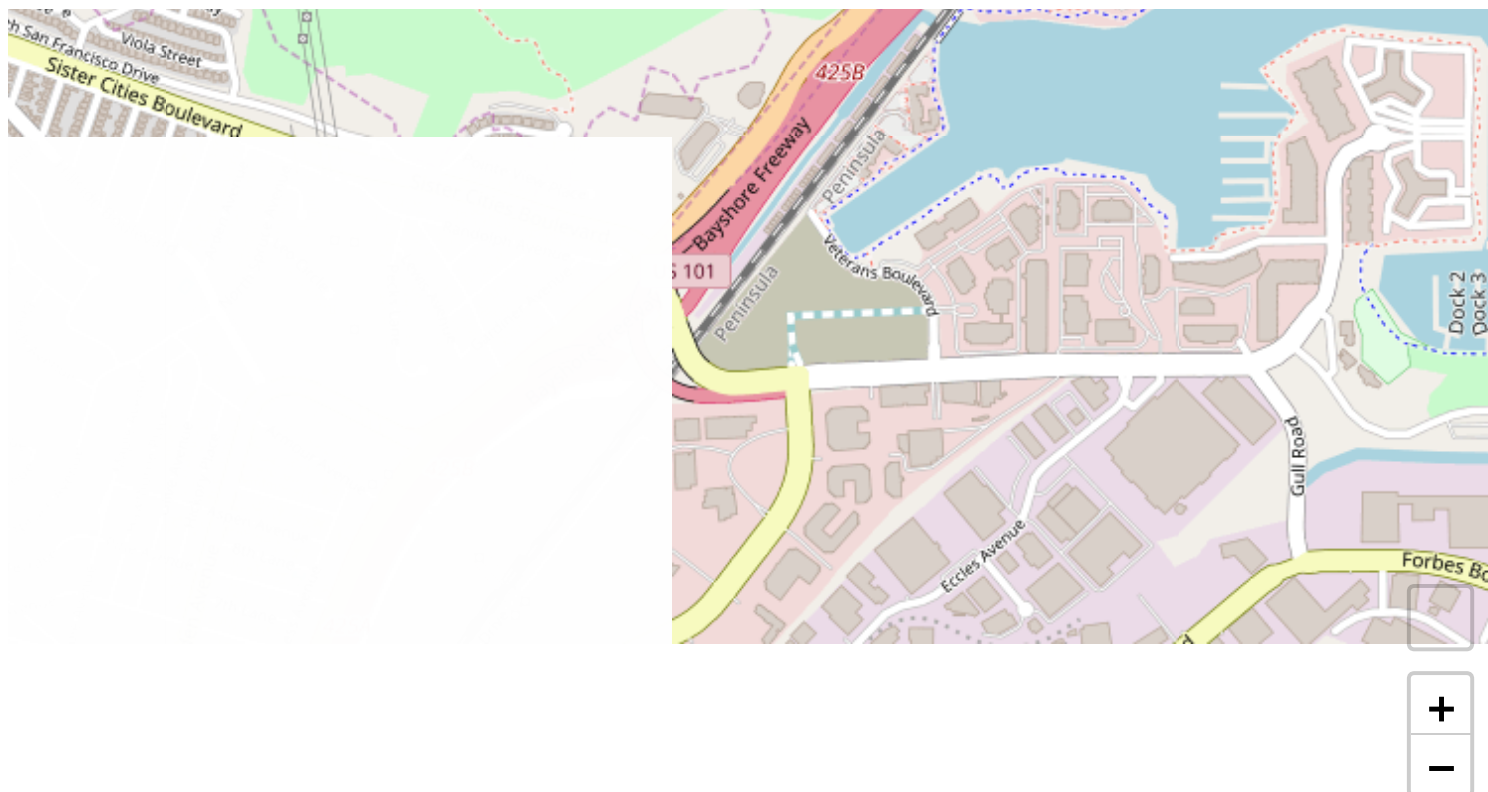
[cartography](#) ([/tag/cartography](#)) [tangram](#) ([/tag/tangram](#)) [documentation](#) ([/tag/documentation](#)) [vector-tiles](#) ([/tag/vector-tiles](#))

Our Mapzen house styles have been updated for v1.0 Mapzen vector tiles. Learn more about our v1.0 vector tile updates in yesterday's **blog post** (<https://mapzen.com/blog/v1-vector-tile-service>).

Along with the latest data updates, we've updated the styles to leverage Tangram's new labeling powers. In the image below you can see the vast improvements on label priority, placement, and positioning.



Explore all our house styles below:



© Mapzen (<https://www.mapzen.com/rights>), OpenStreetMap (<https://openstreetmap.org/copyright>), and others (<https://www.mapzen.com/rights/#services-and-data-sources>) | Leaflet (<http://leafletjs.com/>)

Bubble Wrap

(These maps are interactive! Open full screen ↗ (<https://mapzen.com/products/maps/>))

Get the styles in mapzen.js

Back in July we **announced** (<https://mapzen.com/blog/closing-mapquest-open/>) mapzen.js as an easy way to embed Mapzen house styles into web pages. mapzen.js wraps Leaflet, Tangram, and Mapzen house styles into a general toolkit.

For example:

```
var map = L.Mapzen.map('map', {  
  scene: L.Mapzen.BasemapStyles.Refill  
})
```

See the mapzen.js **Get Started** (<https://mapzen.com/documentation/mapzen-js/get-started/>) guide for full examples.

mapzen.js house styles

The **API reference** (<https://mapzen.com/documentation/mapzen-js/api-reference/#basemap-styles>) lists out all named Mapzen house styles and is currently:

- **Bubble Wrap**
 - `L.Mapzen.BasemapStyles.BubbleWrap`
- **Refill**
 - `L.Mapzen.BasemapStyles.Refill`
 - `L.Mapzen.BasemapStyles.RefillMoreLabels`
 - `L.Mapzen.BasemapStyles.RefillNoLabels`
- **Walkabout**
 - `L.Mapzen.BasemapStyles.Walkabout`
 - `L.Mapzen.BasemapStyles.WalkaboutMoreLabels`
 - `L.Mapzen.BasemapStyles.WalkaboutNoLabels`
- **Tron**
 - `L.Mapzen.BasemapStyles.Tron`
 - `L.Mapzen.BasemapStyles.TronMoreLabels`
 - `L.Mapzen.BasemapStyles.TronNoLabels`
- **Cinnabar**
 - `L.Mapzen.BasemapStyles.Cinnabar`
 - `L.Mapzen.BasemapStyles.CinnabarMoreLabels`
 - `L.Mapzen.BasemapStyles.CinnabarNoLabels`
- **Zinc**
 - `L.Mapzen.BasemapStyles.Zinc`
 - `L.Mapzen.BasemapStyles.ZincMoreLabels`
 - `L.Mapzen.BasemapStyles.ZincNoLabels`

Get src files on the Mapzen CDN

As John alluded to in his **Pools and Polls – Coloring Choropleths** (<https://mapzen.com/blog/coloring-choropleths/>) post last month, it's possible to build data visualizations in Tangram over Mapzen house styles as a basemap (or in map sandwiches).

Tangram's scene import syntax (**documentation** (<https://mapzen.com/documentation/tangram/import/>)) allows one scene file to import another:

```
import: https://mapzen.com/carto/refill-style-no-labels/refill-style-no-labels.yaml
```

Scene files and related assets are available for use in Tangram and Leaflet directly via the Mapzen CDN.

Template

- `https://mapzen.com/carto/{stylename}/{stylefile}.{format}`

Where `{stylename}` and `{stylefile}` are generally the same and will converge over time.

And where `{format}` is mostly `yaml` but Tangram scene file bundles can also be specified as `zip`. Only Tron supports ZIP scene **bundles** (<https://github.com/tangrams/bundler>) at the time of writing.

Mapzen CDN house styles

- **Bubble Wrap**

- `https://mapzen.com/carto/bubble-wrap-style/bubble-wrap.yaml`

- **Refill**

- `https://mapzen.com/carto/refill-style-more-labels/refill-style-more-labels.yaml`
- `https://mapzen.com/carto/refill-style-no-labels/refill-style-no-labels.yaml`
- `https://mapzen.com/carto/refill-style/refill-style.yaml`

- **Walkabout**

- `https://mapzen.com/carto/walkabout-style-more-labels/walkabout-style-more-labels.yaml`
- `https://mapzen.com/carto/walkabout-style-no-labels/walkabout-style-no-labels.yaml`
- `https://mapzen.com/carto/walkabout-style/walkabout-style.yaml`

- **Tron**

- `https://mapzen.com/carto/tron-style/tron-style.zip`
- `https://mapzen.com/carto/tron-style-more-labels/tron-style-more-labels.zip`
- `https://mapzen.com/carto/tron-style-no-labels/tron-style-no-labels.zip`

- **Cinnabar**

- `https://mapzen.com/carto/cinnabar-style-more-labels/cinnabar-style-more-labels.yaml`
- `https://mapzen.com/carto/cinnabar-style-no-labels/cinnabar-style-no-labels.yaml`

- <https://mapzen.com/carto/cinnabar-style/cinnabar-style.yaml>
- **Zinc**
 - <https://mapzen.com/carto/zinc-style-more-labels/zinc-style-more-labels.yaml>
 - <https://mapzen.com/carto/zinc-style-no-labels/zinc-style-no-labels.yaml>
 - <https://mapzen.com/carto/zinc-style/zinc-style.yaml>

Versions

Looking to peg to a specific version of a style? We have you covered!

Template

- <https://mapzen.com/carto/{stylename}/{version}/{stylefile}.{format}>

Where `{version}` is optional and can be major (eg: `1`), major + minor (`1.0`), or major + minor + patch (`1.0.0`).

Partial listing of Refill versions:

- <https://mapzen.com/carto/refill-style/1/refill-style.yaml>
- <https://mapzen.com/carto/refill-style/1.0/refill-style.yaml>
- <https://mapzen.com/carto/refill-style/1.0.0/refill-style.yaml>

Cartography timeline

- **Nov. 2015: A family of styles with many flavours: Introducing Refill, Cinnabar, and Zinc styles for Tangram** (<https://mapzen.com/blog/introducing-refill-cinnabar-and-zinc-styles-for-tangram/>)
- **Mar. 2016: Unboxing Bubble Wrap** (<https://mapzen.com/blog/bubble-wrap-carto/>)
- **Jul. 2016: Walkabout Map Style** (<https://mapzen.com/blog/walkabout/>)
- **Oct. 2016: Tron 2.0 - Creating a Visual Language of Scale** (<https://mapzen.com/blog/tron-2-visual-scale>)
- **Oct. 2016:** Today's house styles update featuring improved Tangram labels and v1.0 vector tile data.

Feedback

Have any feedback on the cartography in Mapzen house styles? Please direct feedback to **Nathaniel Vaughn Kelso** (<https://github.com/nvkelso>), our Chief Cartographer, at **hello@mapzen.com** (<mailto:hello@mapzen.com>)!

· 14 October 2016 ·

**Nathaniel Vaughn Kelso**

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

**Geraldine Sarmiento**

Geraldine is cartographer at Mapzen. She is addicted to shaders.

© 2017 Mapzen

NACIS 2016 roundup

[data](#) ([/tag/data](#)) [whosonfirst](#) ([/tag/whosonfirst](#)) [vector-tiles](#) ([/tag/vector-tiles](#))

If you weren't at NACIS, we missed you! Here are links to the slides of the Mapzen presentations, along with a few teaser screenshots. Video should be available next month. See you next year!

Breaking up with Raster and Going Steady with Vector Tiles

(<https://docs.google.com/presentation/d/17B5sqf2RwDtC9LIYix2G33bm6KafnmwD4Mf-09mGeTU/>) (Practical Cartography Day – Wednesday, 9:00) – *Katie Kowalsky*



breaking up with raster

Cartographer meets map tiles. That infamous meet-cute has caused scores of love, commitment, and eventual heartbreak for all of us in web mapping. The technology behind tiles is constantly changing, growing, and expanding—but where does that leave a cartographer? Are the limits of raster tiles worth abandoning for the **mysterious, bad boy vector tiles** (<https://mapzen.com/projects/vector-tiles/>)? This talk will impart the wisdom of how a cartographer's quest for true love in her tiling scheme and possible workflows can adapt smoothly to a new relationship with vector tiles. ([view slides \(https://docs.google.com/presentation/d/17B5sqf2RwDtC9LIYix2G33bm6KafnmwD4Mf-09mGeTU/edit\)](https://docs.google.com/presentation/d/17B5sqf2RwDtC9LIYix2G33bm6KafnmwD4Mf-09mGeTU/edit))

So thrilled to hear [@KatieKowalsky](https://twitter.com/KatieKowalsky), the Leslie Knope of cartography, on the importance new technologies meeting you at your level.

— vic or treat (@hurricanevicky) **October 19, 2016** (<https://twitter.com/hurricanevicky/status/788774537634586624>)

Who's On First: Administrative Boundaries and Localities
(<https://speakerdeck.com/nvkelso/whos-on-first-administrative-boundaries-and-localities>) (Thursday, 9:00) – *Martin Gamache, Nathaniel Vaughn Kelso*

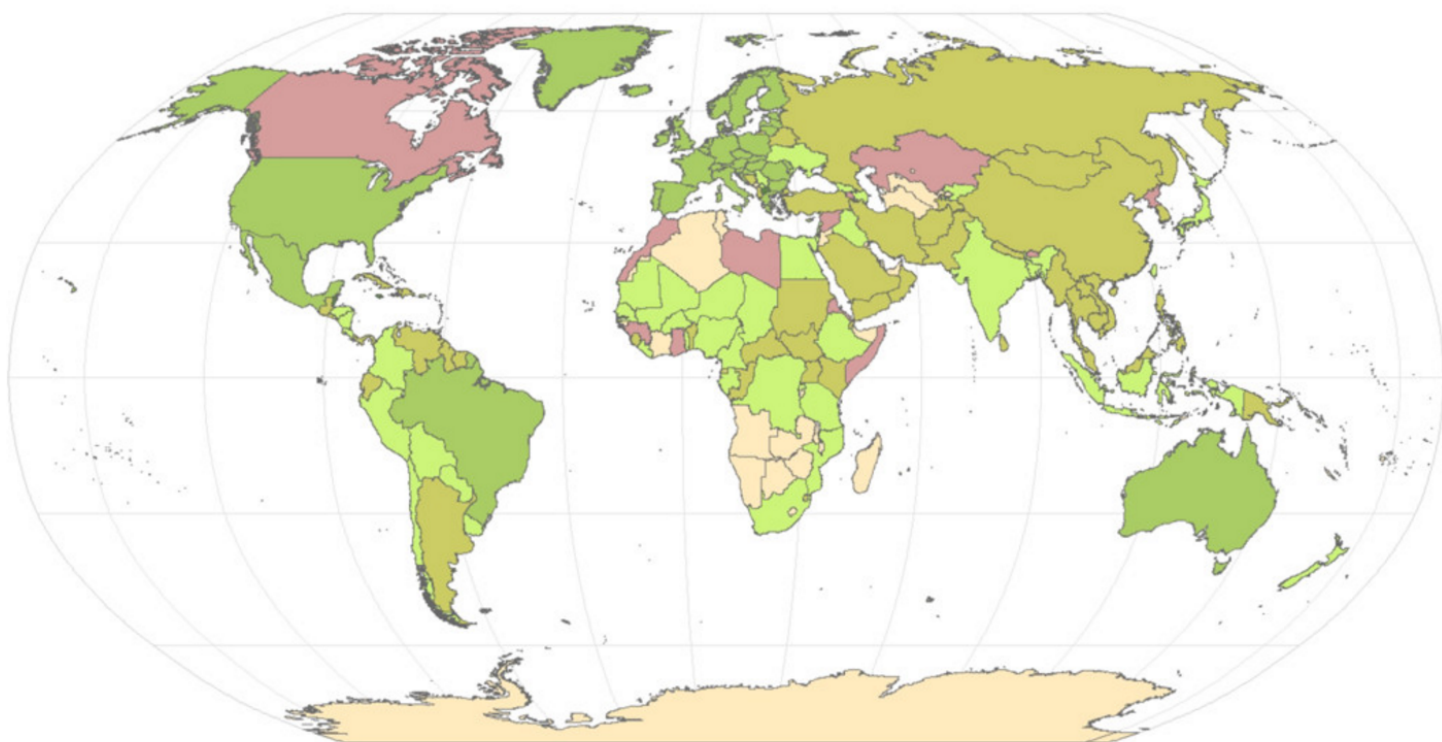
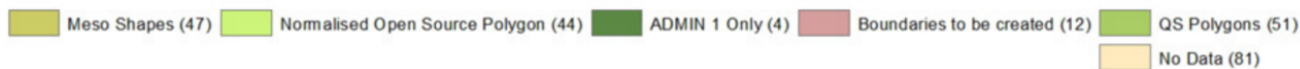
Who's On First (<https://whosonfirst.mapzen.com/spelunker/>) is an open source gazeteer. Administrative level 2 and localities boundaries for countries outside North America and Europe were largely missing from this gazeteer. We have been collaborating on a project to populate or create these from open sources when available or secondary sources when necessary for most of the world. We have built polygons from point sources, scoured the internet for national mapping agency data, and compiled boundaries for dozens of countries around the world, helping create a true open source dataset that can be used for any purpose. ([view slides](#) (<https://speakerdeck.com/nvkelso/whos-on-first-administrative-boundaries-and-localities>))

“If we take the time to closely examine open data efforts, it’s hard to avoid a disheartening conclusion: **most open data isn’t.**”

Jer Thorp

Open for Who? Medium .com Oct. 5, 2016

<https://medium.com/memo-random/open-for-who-ce698a8de79c#.q14rrlo83>

Data Type

Courtney ❄️🎄🎅🎄❄️👋 **Shannon**
@cshannonpdx



I'm at "Drawing the Line" - consistent unique IDs are important
[@mapzen](#) [#nacis2016](#)

7:06 AM · Oct 20, 2016 · Colorado Springs, CO



Who ARE the People in your Neighborhood? Developing Mapzen's Neighborhood Database (<https://speakerdeck.com/nvkelso/who-are-the-people-in-your-neighborhood-developing-mapzens-neighborhood-database>) (Thursday, 9:00) - *Nat Case, Nathaniel Vaughn Kelso*

Mapzen's free and open **Who's On First** (<https://whosonfirst.mapzen.com/spelunker/>) is, like most gazetteers, a big list of places with stable identifiers in a place hierarchy, but we're modeling a new way to think about gazetteers: a space where debate about a place is managed but not decided. Neighborhoods are one of the included place types. In compiling data for this layer, we have encountered challenges in the range of city political structures, the relationship of city to sub-city entities, and in the lack of documentation and data for neighborhoods in many places. Our starting point was shapes derived from Flickr's neighborhood tags, but we discovered additional serious issues with the original compilation of those tags. We'll approach (and not necessarily resolve) these questions in a tour through cities big and small in the United States and around the world showing challenges faced and the final result on the map and in search. ([view slides](https://speakerdeck.com/nvkelso/who-are-the-people-in-your-neighborhood-developing-mapzens-neighborhood-database)) (<https://speakerdeck.com/nvkelso/who-are-the-people-in-your-neighborhood-developing-mapzens-neighborhood-database>)

1. WOE's source data was bad
2. WOE's compilation was bad
3. Flickr's users were bad
4. Quattroshapes and Zetashapes logic was bad
5. Cities are stupid
6. Cities don't have neighborhoods
7. Cities are confusing

There are two basic kinds of populated place geography:
radial and bounded.

Every named populated place has at least the potential
to have both kinds of geography.

Who's On First overview (<https://speakerdeck.com/nvkelso/whos-on-first-overview>)– *Nathaniel Vaughn Kelso* (Thursday, 9:00)

One Minute Map

mapzen-js (/tag/mapzen-js) **demo** (/tag/demo) **tutorial** (/tag/tutorial)

tangram (/tag/tangram) **make-your-own** (/tag/make-your-own)

Back in July, **we mentioned that we were testing a new library** (<https://mapzen.com/blog/closing-mapquest-open/>) to simplify the process of displaying Mapzen's signature maps (like **Bubble Wrap** (<https://mapzen.com/blog/bubble-wrap-carto/>), **Walkabout** (<https://mapzen.com/blog/walkabout/>), and **Tron 2.0** (<https://mapzen.com/blog/tron-v2-visual-scale/>)) on the web. Today we are starting a new series looking at how you can use **mapzen.js** (<https://github.com/mapzen/mapzen.js>) to create and embed beautiful maps on your website.

As an introduction to mapzen.js, I want to walk you through how to create a map on the web in under one minute. Just a simple map, but a map with a link that you can send to anyone with an internet connection. **In. Just. One. Minute.**

After that, we'll take a look at some of the bells and whistles that mapzen.js provides, to help give your map that little something extra.

The One Minute Map

Let's start by setting up a simple map. We could do this by creating an HTML file right on your local computer, and we will do that later in the series. But for this exercise, I'm going to show you how to use **GitHub gists** (<https://help.github.com/articles/about-gists/>) to create a map in no time at all (and no GitHub account required).

Set your timer and let's begin...

1. Go to (<https://gist.github.com>)<https://gist.github.com> (<https://gist.github.com>).
If you have a GitHub account, go ahead and log in, but you don't need one for this exercise.
2. For "Filename including extension" type in `index.html` . (*Gist description is optional.*)

3. In the large white box, copy and paste in the following html code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>One Minute Map</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="https://mapzen.com/js/mapzen.css">
    <script src="https://mapzen.com/js/mapzen.min.js"></script>
    <style>
      #map {
        height: 100%;
        width: 100%;
        position: absolute;
      }
      html,body{margin: 0; padding: 0}
    </style>
  </head>
  <body>
    <div id="map"></div>
    <script>
      // Mapzen API key (replace key with your own)
      // To generate your own key, go to https://mapzen.com/developers/
      L.Mapzen.apiKey = 'mapzen-JA21Wes';

      var map = L.Mapzen.map('map', {
        center: [37.7749, -122.4194],
        zoom: 13,
        scene: L.Mapzen.BasemapStyles.Cinnabar
      });

    </script>
  </body>
</html>
```

4. Click **Create public gist**. The page will reload with your saved file (i.e., your *gist*).

5. Next, click into the URL at the top of your browser and replace “gist.github.com” with “bl.ocks.org”

so that: **https://gist.github.com/<username>/<id>**

becomes: **https://bl.ocks.org/<username>/<id>**

6. Hit enter to view your map at **https://bl.ocks.org/<username>/<id>**

Stop that timer because you just made a map! In under a minute!

w00t! Go ahead and take a look around. Click “Open” to view the map full screen. Click and drag to move around. Click the plus and minus buttons to zoom in and out.

I'll wait here while you explore. Here's a handy countdown GIF (no pressure!) and a screenshot of what you should see in the gist:



GitHub Gist Search... All gists GitHub Sign up for a GitHub account Sign in

Instantly share code, notes, and snippets.

Gist description...

index.html type here paste here Spaces 2 No wrap

```

11     width: 100%;
12     position: absolute;
13   }
14   html,body{margin: 0; padding: 0}
15 </style>
16 </head>
17 <body>
18   <div id="map"></div>
19   <script>
20
21     var map = L.Mapzen.map('map', {
22       center: [37.7749, -122.4194],
23       zoom: 13,
24       scene: L.Mapzen.BasemapStyles.Cinnabar
25     });
26
27   </script>
28 </body>
29 </html>

```

Add file Create secret gist Create public gist click here

Under The Hood

Pretty cool, right? Now let's take a closer look at this file and some of the options we have available.

Note: If you've signed into GitHub, you'll be able to edit your existing gist. If you are working with an "anonymous" gist, you'll need to create a new one at <https://gist.github.com> (<https://gist.github.com>).

To create this map, we needed three essential pieces:

1. The reference to our mapzen.js library, which includes a JavaScript file and a CSS file:

```

<link rel="stylesheet" href="https://mapzen.com/js/mapzen.css">
<script src="https://mapzen.com/js/mapzen.min.js"></script>

```

2. An html div container for our map:

```
<div id="map"></div>
```

3. Some simple JavaScript to define our map and attach it to the `map` div:

```
// Mapzen API key (replace key with your own)
// To generate your own key, go to https://mapzen.com/developers/
L.Mapzen.apiKey = 'mapzen-JA21Wes';

var map = L.Mapzen.map('map', {
  center: [37.7749, -122.4194],
  zoom: 13,
  scene: L.Mapzen.BasemapStyles.Cinnabar
});
```

UPDATE March 1, 2017: A Mapzen developer API key is now required for mapzen.js. We've updated the Make Your Own series to include a demo key. Generate your own free API key at <https://mapzen.com/developers/> (<https://mapzen.com/developers/>).

In this snippet (above), we set the starting point of the map at San Francisco ([37.7749, -122.4194]), with a zoom level of 13. The `scene` is one of Mapzen's house styles, **Cinnabar** (<https://mapzen.com/blog/introducing-refill-cinnabar-and-zinc-styles-for-tangram/>).

Try swapping out `L.Mapzen.BasemapStyles.Cinnabar` with one of our other house styles: <https://mapzen.com/documentation/mapzen-js/api-reference/#basemap-styles> (<https://mapzen.com/documentation/mapzen-js/api-reference/#basemap-styles>)

For example:

```
var map = L.Mapzen.map('map', {
  center: [37.7749, -122.4194],
  zoom: 13,
  scene: L.Mapzen.BasemapStyles.Refill
});
```

You may have noticed from the **API documentation** (<https://mapzen.com/documentation/mapzen-js/api-reference/#basemap-styles>) that each of these constants is simply a reference to a specific `.yaml` file. In this case, `L.Mapzen.BasemapStyles.Refill` points to `https://mapzen.com/carto/refill-style/refill-style.yaml`.

YAML (<http://www.yaml.org/>) (a delightfully recursive acronym that stands for *YAML Ain't Markup Language*) is the format expected by **Tangram** (<https://mapzen.com/products/tangram/>), the map rendering engine used by mapzen.js to render Mapzen's house basemap styles.

Next week, I'll show you how we can create a scene file to style our own data.

Bells and Whistles

Up to this point, we could have used **Leaflet** (<http://leafletjs.com/>) + **Tangram's own javascript API** (<https://mapzen.com/documentation/tangram/javascript-API/>) to create the map above. Why use mapzen.js at all?

We've built mapzen.js to be an easy entry point into Mapzen maps and services. In addition to quickly adding **Mapzen map styles** (<https://mapzen.com/products/maps/>) to a Leaflet map, we've added a few "bells and whistles" to help you test out some of our other services.

Let's dive back into our `index.html` file and check out a few of these services.

Again, edit your gist or create a new one to follow along.

Search

How about a search box so you can jump to your favorite locations? At the bottom of your script block, add the following:

```
var geocoder = L.Mapzen.geocoder();
geocoder.setPosition('topright');
geocoder.addTo(map);
```

The `topright` position refers to the map corner we wish to display this control. These options are part of **Leaflet's Control Positions** (<http://leafletjs.com/reference.html#control-positions>).

See the **geocoder API reference** (<https://mapzen.com/documentation/mapzen-js/search/>) for more information on customizing the interaction and search query behavior.

Locator

Let's also add a locator button so you can quickly zoom to your own location:

```
var locator = L.Mapzen.locator();  
locator.addTo(map);
```

(Note: you'll need allow your browser to share your location for this to work.)

URL Hash

And finally, we'll add a URL hash to the map, to allow for **deep linking** (https://en.wikipedia.org/wiki/Deep_linking) to the map location and zoom level.

```
L.Mapzen.hash({  
  map: map,  
  geocoder: geocoder  
});
```

As you zoom and pan around the map, the position will be saved as a URL hash, which allows you to send a link to specific place on your map.

Your `index.html` file should now look something like this:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Bells and Whistles</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="https://mapzen.com/js/mapzen.css">
    <script src="https://mapzen.com/js/mapzen.min.js"></script>
    <style>
      #map {
        height: 100%;
        width: 100%;
        position: absolute;
      }
      html,body{margin: 0; padding: 0}
    </style>
  </head>
  <body>
    <div id="map"></div>
    <script>
      // Mapzen API key (replace key with your own)
      // To generate your own key, go to https://mapzen.com/developers/
      L.Mapzen.apiKey = 'mapzen-JA21Wes';

      var map = L.Mapzen.map('map', {
        center: [37.7749, -122.4194],
        zoom: 13,
        scene: L.Mapzen.BasemapStyles.Refill
      });

      // Mapzen search box
      var geocoder = L.Mapzen.geocoder();
      geocoder.setPosition('topright');
      geocoder.addTo(map);

      // Mapzen locator (i.e., Find my location)
      var locator = L.Mapzen.locator();
      locator.addTo(map);

      // Mapzen hash (for deep linking to map location and state)
      L.Mapzen.hash({
        map: map,
        geocoder: geocoder
      });

    </script>
  </body>
</html>
```

Full demo can be seen at

(<https://bl.ocks.org/rfriberg/3c1e2447a77958e893a74a279014499b>)

(<https://bl.ocks.org/rfriberg/3c1e2447a77958e893a74a279014499b>)

Thanks for coming along on this introductory tour of mapzen.js. Today was just a small taste of all the great things going into mapzen.js and the power behind the Tangram rendering engine. Next week, we'll dive into how we can set up a scene file to display our own data over one of Mapzen's signature basemaps.

In the meantime, if you have any questions or want to show us how you're using mapzen.js, just **drop us a line (<mailto:hello@mapzen.com>)!**

~~~

Check out additional tutorials from the **Make Your Own (<https://mapzen.com/tag/make-your-own/>)** series:

- **One Minute Map (<https://mapzen.com/blog/one-minute-map/>)**
- **Map Sandwich (<https://mapzen.com/blog/map-sandwich/>)**
- **Filters & Functions (<https://mapzen.com/blog/filters-and-functions/>)**
- **Put A Label On It (<https://mapzen.com/blog/tangram-labels/>)**
- **Interactive Mapping with Tangram (<https://mapzen.com/blog/tangram-interactivity/>)**
- **Lots of Dots (<https://mapzen.com/blog/lots-of-dots/>)**
- **Customize Your Search (<https://mapzen.com/blog/customize-your-search/>)**

· 26 October 2016 ·



**Rhonda Friberg**

Rhonda is a javascripter who makes maps, trouble, and the occasional pie.

© 2017 Mapzen



# नक्शे! नक्शे! मानचित्र! நிலப்படம்! नक्ष! नकाशे! Maps!

tangram (/tag/tangram)

मॅप्स एक क्षेत्र की कहानी और इतिहास को दर्शाता है। और यह आपका हक है की आप अपने क्षेत्र की भाषा में मॅप्स का उपयोग कर सकें। आप मपज़ें और टांगराम के ज़रिए हिन्दी और भारत की अन्य भाषाओं में मॅप्स की रचना कर सकते हैं।

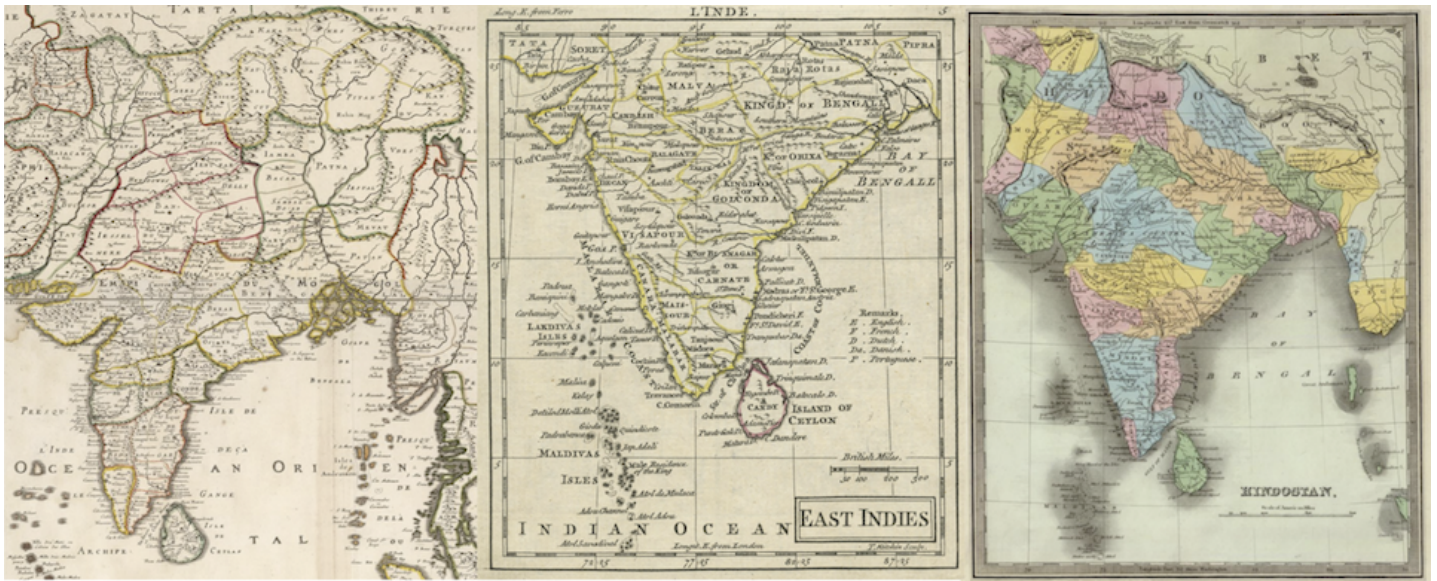
মানচিত্র আমাদের ইতিহাস এবং গল্পের ছিন্তা. আর মানচিত্র নিজেদের ভাসএ ব্যবহার করার অধিকার আপনার সবার. MAPZEN আর TANGRAM ব্যবহার করে বাংলাএ এবং অন্যান্য ভারতীয়া ভাসএ মানচিত্র সৃষ্টি করতে পারেন.

مپس ایک علاقے کی کہانی اور تاریخ کی عکاسی کرتا ہے. اور یہ آپ حق ہے کہ آپ اپنے علاقے کی زبان میں مپس کو استعمال کر سکیں. آپ مپڈے اور ٹاگرام کے ذریعہ ہندی اور ہندوستان کی دیگر بھاشاؤں میں مپس کی تحریر کر سکتے ہیں

ನಕ್ಷೆಗಳು ಪ್ರಾದೇಶಿಕ ಹಾಗೂ ಇತಿಹಾಸ ಕತೆಗಳ ಸಂಕೇತವಾಗಿದೆ. ನಿಮ್ಮ ಮಾತೃ ಭಾಷೆಯಲ್ಲಿ ನಕ್ಷೆಗಳನ್ನು ಉಪಯೋಗಿಸುವ ಹಕ್ಕು ನಿಮ್ಮದಾಗಿದೆ. ನಿಮ್ಮ ಭಾಷೆಯಲ್ಲಿ ನಕ್ಷೆಗಳನ್ನು ಸೃಷ್ಟಿ ಮಾಡಲು ಮ್ಯಾಪ್‌ಝೆನ್ ಮತ್ತು ಟ್ಯಾನ್ ಗ್ರಾಮನ್ನು ಬಳಕೆ ಮಾಡಬಹುದಾಗಿದೆ.

As we note in Hindi, Bengali, Urdu and Kannada above, maps tell the story of a region, and at Mapzen we think you should be able to tell these stories in your own language. Wherever you live, you can use Mapzen and Tangram to localize your maps.

India has many stories to tell, from the mythological world to years of dynastic and colonial rule. Each time that lines on the maps move, there is a new story to tell, **even today** (<http://timesofindia.indiatimes.com/india/15-facts-you-need-to-know-about-Telangana/articleshow/35955351.cms>).



Images via **David Rumsey** (<http://www.davidrumsey.com/>). Left: *L'Inde deca et dela le Gange, ou est l'Empire du Grand Mogul* (<http://www.davidrumsey.com/luna/servlet/detail/RUMSEY~8~1~280875~90053713>), and *Partie Meridionale de l'Inde En deux Presque Isles l'une de ca et l'autre de la le Gange* (<http://www.davidrumsey.com/luna/servlet/detail/RUMSEY~8~1~280876~9005371>) (Nicolas and Guillaume Sanson, 1697/1703/1709). Center: *A New General and Universal Atlas Containing Forty five Maps By Andrew Dury* (<http://www.davidrumsey.com/luna/servlet/detail/RUMSEY~8~1~285652~90058167>) (Andrew Dury, 1763). Right: Hindostan, p. 106 from *A New Universal Atlas; Comprising Separate Maps Of all the Principal Empires, Kingdoms & States Throughout the World* (<http://www.davidrumsey.com/luna/servlet/detail/RUMSEY~8~1~3289~400106>) (David Burr & Thomas Illman, 1835)

In 2014, India's Commissioner for Linguistic Minorities (<http://nclm.nic.in/shared/linkimages/NCLM50thReport.pdf>) talked about the ties between language and culture:

India, one of the world's ancient civilizations, puts forth a magnificent mosaic of multiple castes, religions and languages. Our centuries-old multilingual, multicultural ethos has held the country together like the thread in the rosary of beads, representing 'unity in diversity' in our country.

There are 23 official languages in India (including English). According to the **2001 Census language statistics** ([http://www.censusindia.gov.in/Census\\_Data\\_2001/Census\\_Data\\_Online/Language/Statement4.aspx](http://www.censusindia.gov.in/Census_Data_2001/Census_Data_Online/Language/Statement4.aspx)) (the 2011 Census language data is not yet available). While Hindi and Bengali are the

most widely spoken languages, **other languages are the majority in many Indian states** ([http://www.censusindia.gov.in/Census\\_Data\\_2001/Census\\_Data\\_Online/Language/part4.htm](http://www.censusindia.gov.in/Census_Data_2001/Census_Data_Online/Language/part4.htm)).

| Language | Native Speakers | Majority States |
|----------|-----------------|-----------------|
| Hindi    | 422 million     | 10              |
| Bengali  | 83 million      | 7               |
| Telugu   | 74 million      | 1               |
| Marathi  | 72 million      | 1               |
| Tamil    | 60 million      | 2               |
| Urdu     | 52 million      | 0               |
| Gujarati | 46 million      | 2               |
| Kannada  | 37 million      | 1               |

Below is a map of Indian states, with the name in the majority language.



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

*Open full screen! (<https://mapzen.com/tangram/view/?scene=https://mapzen-assets.s3.amazonaws.com/resources/languages-of-india.yaml#6.459/22.796/79.027#6/21.197/82.66>)*

The number of people in each Indian state rivals that of many nations. Population data is available in the Mapzen gazetteer **Who's On First** (<https://whosonfirst.mapzen.com/spelunker/>), and by **exporting the states of India as a GeoJSON feature collection** ([https://burritojustice.github.io/wof-descender/?wof\\_id=85632469&wof\\_level=region](https://burritojustice.github.io/wof-descender/?wof_id=85632469&wof_level=region)) we were able to import it into Tangram and display it below the native language names, along with a greyscale choropleth and a dynamic comparison to the population of a certain “large” American state.



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

*Open full screen! (<https://tangrams.github.io/tangram-frame/?lib=0.10&url=https://mapzen-assets.s3.amazonaws.com/resources/languages-of-india-california.yaml#6/24.602/79.464>)*

(Note that we are overlaying the state boundaries and labels on the **Mapzen house style** (<https://mapzen.com/blog/updated-house-styles/>), Refill. While we could have picked the `no-labels` version, we chose to fade out Refill's labels. This way, cities are still visible for context in the greyscale choropleth style, but the state names and data we are displaying have priority. You can see what's happening **behind the scenes in Tangram Play** (<https://mapzen.com/tangram/play/?scene=https://mapzen.com/api/scenes/22/17/resources/scene.yaml#5.013/21.415/78.694>). If you want to make your own population comparison, change `global.population` and `global.compare_nation`. You can also pick a variety of base colors for the choropleth by altering `choropleth_color` in the `global` variable section.)

Users around the world have added hundreds of languages with different scripts to places in OpenStreetMap. The Tangram mapping engine is flexible enough to support most of these, including those that render with text shaping and text positioning, such as हिन्दी (Hindi), ಕನ್ನಡ (Kannada), தமிழ் (Tamil), ગુજરાતી (Gujarati), and తెలుగు (Telugu). It can also properly display right-to-left languages like اردو (Urdu) (which is also rendered with text shaping and positioning).

Our browser-based **Tangram JS** (<https://github.com/tangrams/tangram>) renderer utilizes Canvas to render these scripts. The **Tangram ES** (<https://github.com/tangrams/tangram-es>) native renderer uses **Alfons** (<https://github.com/hjanetzek/alfons>), an amalgamation of some very useful open-source text rendering libraries like **HarfBuzz** (<https://www.freedesktop.org/wiki/Software/HarfBuzz/>), **Freetype** (<https://www.freetype.org/>), and **ICU** (<http://site.icu-project.org/>) to serve vertices and glyph textures for our OpenGL renderer.

But how to access these languages? We are adding language flags to Mapzen in-house styles which will allow you to display your desired language with very little effort. It's already available in Bubble Wrap (and is coming to the rest of our house styles soon!) Below you can see maps of Karnataka with text labels in Kannada.



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

Open full screen! (<https://tangrams.github.io/tangram-frame/?lib=0.10&url=https://s3.amazonaws.com/mapzen-assets/images/languages-of-india/kannada.yaml#9.2880/17.3224/78.0438>)

To change the text, import the standard Bubble Wrap default scene file and modify the language global value to your preferred **language code**

([http://wiki.openstreetmap.org/wiki/Wiki\\_Translation#Language\\_prefixes\\_and\\_namespaces](http://wiki.openstreetmap.org/wiki/Wiki_Translation#Language_prefixes_and_namespaces)).

```
import: https://mapzen.com/cartto/bubble-wrap-style/bubble-wrap.yaml

global:
  ux_language: kn
```

(The language value can also be changed via the **Tangram JavaScript API** (<https://github.com/tangrams/bubble-wrap/blob/37259a255dc725f13885b79687f27ac6ba1c7c78/main.js#L222-L223>).)

Where do these labels come from? We are able to show maps in Kannada because residents of Karnataka and native speakers of Kannada have been hard at work **adding over 40,000 labels to nodes, ways and relations in OpenStreetMap** (<https://thejeshgn.com/projects/state-of-indian-languages-openstreetmap/>).

Changing the two letter language code gets you a map of Maharashtra with Marathi labels:

```
import: https://mapzen.com/cartto/bubble-wrap-style/bubble-wrap.yaml

global:
  ux_language: mr
```





This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

*Open full screen! (<https://tangrams.github.io/tangram-frame/?lib=0.10&url=https://s3.amazonaws.com/mapzen-assets/images/languages-of-india/marathi.yaml#9.2880/16.6858/74.4642>)*

While there are only 5,000 tags in Marathi, you can make this better in OpenStreetMap! Thejesh GN has been doing excellent work **tracking the progress of Indian native language maps** (<http://thejeshgn.com/2016/09/30/where-are-the-indian-language-maps/>) and has a **tutorial on how to add translations to OSM** (<https://thejeshgn.com/wiki/notebook/translating-openstreetmap-to-kannada-or-any-language/>).

Whether you speak one of India's 22 official languages, or one of the hundreds of others spoken on the Indian subcontinent, or any other language in the world, add it to OpenStreetMap! **Mapzen Vector Tiles** (<https://mapzen.com/blog/v1-vector-tile-service/>) are updated constantly, and any tags you add in your native language will show up quickly, usually within a day! Make your map tell a story in your mother tongue!

*Disputed Territories between India and its neighbouring countries: The dashed lines in the map represent the disputed/claimed boundaries between India-Pakistan and India-China. The map is in accordance with the **UN representation of the Indian border** (<http://www.un.org/Depts/Cartographic/map/profile/Souteast-Asia.pdf>) and the shape of the entire Indian map including the dashed lines is in accordance **with Indian Law** ([http://www.surveyofindia.gov.in/files/Correct\\_India\\_Map\\_1.pdf](http://www.surveyofindia.gov.in/files/Correct_India_Map_1.pdf)).*

· 30 October 2016 ·



**Varun Talwar**

Graphics Engineer. C++, Graphics and Physics Simulation enthusiast. Worked on Animation Movies.



**John Oram**

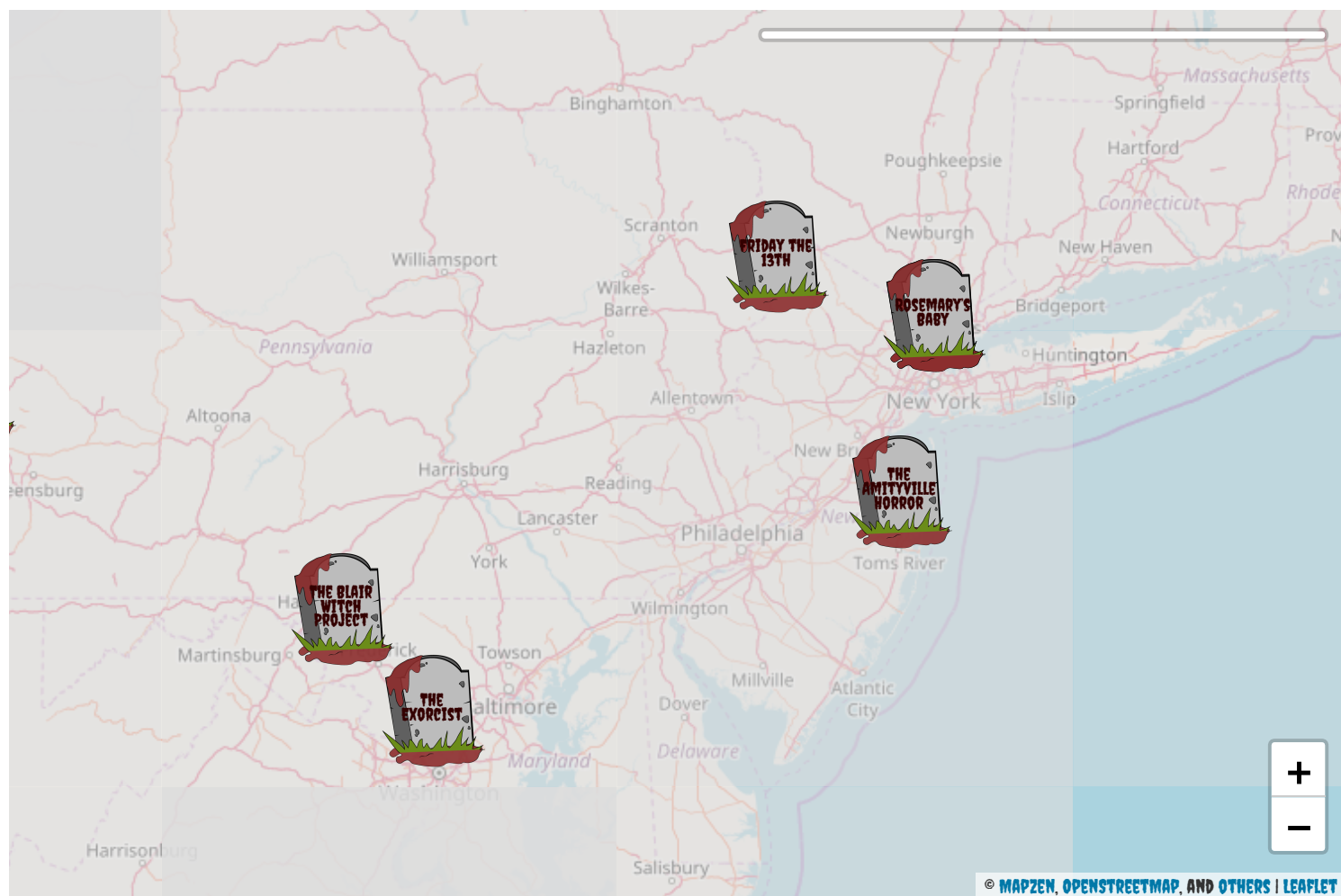
Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

---

© 2017 Mapzen

# A Halloween road trip

It's Halloween and what better way to celebrate than a spoOOOoOOOoky road trip? From *Amityville Horror* to *The Undead*, we're taking a trip across the continental United States to visit all of the locations these horror movies were filmed. This schlep won't be a nightmare, since we're creating the route using **Mapzen Turn-by-Turn** (<https://mapzen.com/documentation/mobility/optimized/api-reference/>).



This map is interactive! View full screen ↗ (<https://mapzen.github.io/halloween2016/>)

Making this ghoulish map wasn't a fright thanks to **Mapzen.js** (<https://mapzen.com/documentation/mapzen-js/>), **Tangram** (<https://mapzen.com/documentation/tangram/>), and **Mapzen Turn-by-Turn** (<https://github.com/mapzen/lrm-mapzen>). We started with a list of our list of favorite horror movies (we may or may not have visited some of them 😊) and locations associated with them

that a fan could visit. For example, the M street steps where Father Karras plummets down to his death have been since renamed “The Exorcist Steps” and going up and down them is a great leg workout! We also discovered **this gem** (<http://www.movie-locations.com/>), listing where some of our favorite moments in horror movie history take place. Movie mappers rejoice! We then geocoded the locations and added details like location type to an **array** (<https://github.com/mapzen/halloween2016/blob/master/js/mapzen-horror-road-trip.js>) so we knew what to look for when we drove there.

To design the basemap, we played around with **kinkade** (<http://tangrams.github.io/kinkade/>) to generate a **sphere map** (<https://github.com/mapzen/halloween2016/blob/gh-pages/img/sphere-map2.png>) based off of **Geraldine’s experiments** (<https://mapzen.com/blog/sphere-maps/>) with terrain. After creating the terrain, we used the **waves** (<http://tangrams.github.io/blocks/#patterns-waves>) and **stripes** (<http://tangrams.github.io/blocks/#patterns-stripes>) block to add some detail to the water and landuse features. The **‘Creepster’ typeface** (<https://fonts.google.com/specimen/Creepster>) was the perfect addition to this map with autumnal reds and oranges added to pop against the dark and “scary” terrain. A regular pin marker won’t suffice for this scary map, so we designed custom icons for the movie locations and the cursor. We love being able to add dynamic data with Tangram, so we animated the route we’re taking, from the apartment building in Rosemary’s Baby to the Amityville Horror house.

Explore the map and be sure to click on the different movie list titles for a scary surprise!

*Witches image via **Flickr user mryantaylor***

*(<https://www.flickr.com/photos/mryantaylor/3877617646>).*

· 31 October 2016 ·



### Katie Kowalsky

Katie is a mapslinger who knows how to make a proper cheese plate and is a serial to-do list maker.



### Ekta Daryanani

Lead, Design + User Experience. Thinks: colors, people, community and city life. Does: mobile, maps, rock climbing and yoga.

### Hanbyul Jo



Hanbyul does front-end that makes maps and candies.

---

© 2017 Mapzen

# Map Sandwich

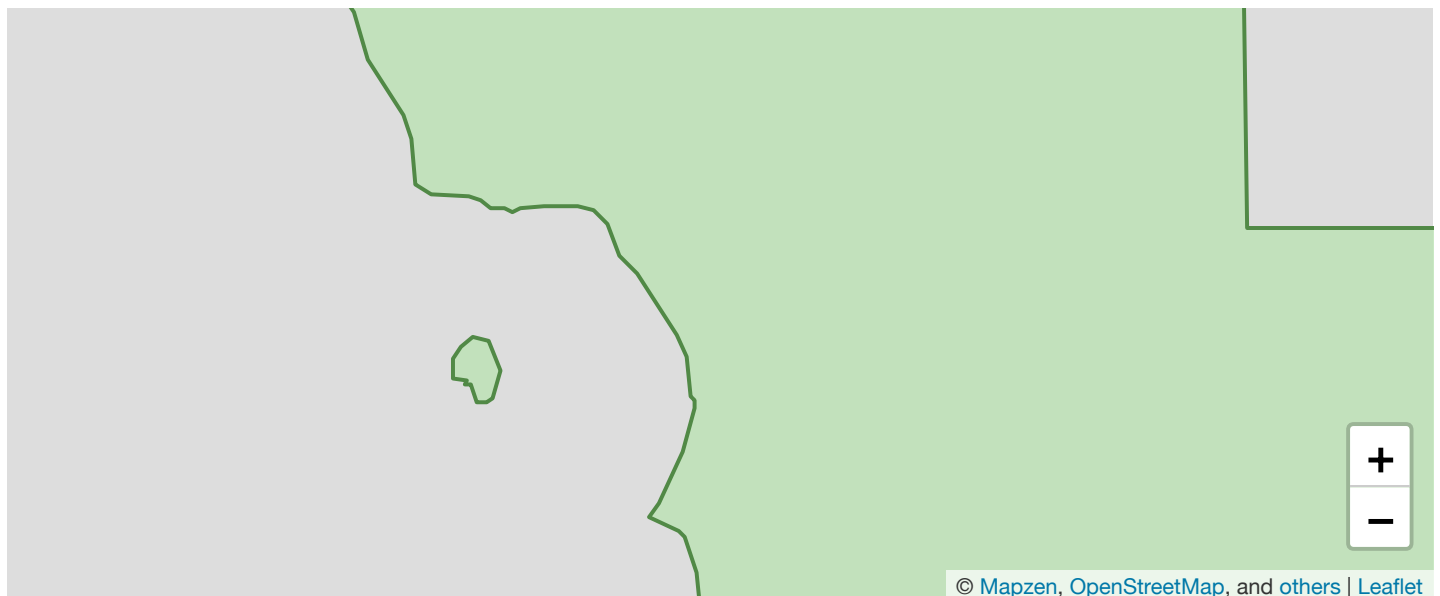
**mapzen-js** (/tag/mapzen-js) **demo** (/tag/demo) **tutorial** (/tag/tutorial)

**tangram** (/tag/tangram) **make-your-own** (/tag/make-your-own)

Welcome back! In last week's post, **One Minute Map** (<https://mapzen.com/blog/one-minute-map/>), I walked you through how to get a simple map up on the web in just one minute using **mapzen.js** (<https://mapzen.com/documentation/mapzen-js/>). This week, as promised, I'm here to show you how we can expand that simple map to display and style our own data. We'll continue using **mapzen.js** (<https://mapzen.com/documentation/mapzen-js/>) for our basic map and will use a **Tangram** (<https://mapzen.com/documentation/tangram/>) scene file to display our data on that map.

But why bother with a Tangram scene file? Isn't **mapzen.js** (<https://mapzen.com/documentation/mapzen-js/>) (and **tangram.js** (<https://mapzen.com/documentation/tangram/>)) just an extension of **Leaflet** (<http://leafletjs.com/>)? Can't you just add a data overlay using Leaflet's built-in methods?

You can! And it's a completely acceptable way of adding data to a map. Here's a quick demo to show you how this might work:



**View demo on bl.ocks.org** (<https://bl.ocks.org/rfriberg/9b85b62bf26f4a562bf09879b4a4354b>)

However.... notice how the polygons cover up the map labels? If you've made a web map in the past few years, you are probably familiar with this pattern. There are a lot of opinions out there about obscuring basemap labels, but let's just say... *it's less than ideal*.

It's not quite the **map sandwich** (<https://blogs.esri.com/esri/arcgis/2009/07/13/the-map-sandwich/>) to which we aspire.



*The classic map sandwich.*

We can do better. In fact, let's get started, and I'll show you how to make your very own map sandwich.





Last week, we used a very (ahem) fancy <https://gist.github.com> (<https://gist.github.com>) to <https://bl.ocks.org> (<https://bl.ocks.org>) conversion to get our map on the web. It's great for publishing a map very quickly, but not quite as great for making lots of little tweaks to our code as we test out new things.

If you're ready to explore an alternative workflow, we've written some docs on **setting up a bare bones development environment** (<https://mapzen.com/documentation/guides>). This includes setting up a text editor and a few options for getting your map up on a web server. If you're new to web development, I recommend you **give it a read** (<https://mapzen.com/documentation/guides>).

If you'd rather skip the text editor, go ahead and use the [gist.github.com](https://gist.github.com) (<https://gist.github.com>) workflow from **last week** (<https://mapzen.com/blog/one-minute-map/>) or check out [blockbuilder.org](http://blockbuilder.org/) (<http://blockbuilder.org/>), which automates the gist → bl.ocks workflow for you.


*Thanks for the tip @ericsoco!*

**ericsoco**  
@ericsoco







Thx for the One Minute Map, @mapzen! [mapzen.com/blog/one-minute-map/](https://mapzen.com/blog/one-minute-map/) FYI, you can make it a 5-second map with [blockbuilder.org](http://blockbuilder.org/) /cc @enjalot

12:22 PM - Oct 26, 2016



**One Minute Map**  
An introductory tour of mapzen.js  
[mapzen.com](https://mapzen.com)

 1  2  7 

OK, time to make a map...

We'll start with a basic mapzen.js web map. Open up your text editor (or **[gist.github.com](https://gist.github.com)** (**<https://gist.github.com>**)) and create a new file called `index.html`. Copy and paste in the following code block. It should look similar to what we used last week.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>San Juan Island</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="https://mapzen.com/js/mapzen.css">
    <script src="https://mapzen.com/js/mapzen.min.js"></script>
    <style>
      #map {
        height: 100%;
        width: 100%;
        position: absolute;
      }
      html,body{margin: 0; padding: 0}
    </style>
  </head>
  <body>
    <div id="map"></div>
    <script>
      // Mapzen API key (replace key with your own)
      // To generate your own key, go to https://mapzen.com/developers/
      L.Mapzen.apiKey = 'mapzen-JA21Wes';

      var san_juan_island = [48.5326, -123.0879];

      var map = L.Mapzen.map('map', {
        center: san_juan_island,
        zoom: 11,
        scene: L.Mapzen.BasemapStyles.Refill
      });

      // Move zoom control to the top right corner of the map
      map.zoomControl.setPosition('topright');

      // Mapzen Search box
      const geocoder = L.Mapzen.geocoder('mapzen-JA21Wes');
      geocoder.addTo(map);

    </script>
  </body>
</html>
```

*UPDATE March 1, 2017: A Mapzen developer API key is now required for mapzen.js. We've updated the Make Your Own series to include a demo key. Generate your own free API key at <https://mapzen.com/developers/> (<https://mapzen.com/developers/>).*

# Anatomy of a Scene File

Take a close look at the `scene` parameter:

```
scene: L.Mapzen.BasemapStyles.Refill
```

As I hinted last week, we can replace `L.Mapzen.BasemapStyles.Refill` with the path to our very own scene file. Go ahead and update that line to:

```
scene: 'scene.yaml'
```

Now let's create that file. In the same directory as `index.html`, create a new file called `scene.yaml`. (If you're working with a gist, be sure to add the `scene.yaml` file to the same gist using the "Add file" button at the bottom.) Copy and paste in the following code:

```
import: https://mapzen.com/carto/refill-style/6/refill-style.yaml

sources:
  _nps_boundary:
    type: GeoJSON
    url: https://gist.githubusercontent.com/rfriberg/684645c22f495b4a46f29fb312b6d268,

layers:
  _national_park:
    data: { source: _nps_boundary }
    draw:
      ...
```

So, what are we looking at here? There is a lot of information about scene files in **Tangram's documentation** (<https://mapzen.com/documentation/tangram/Scene-file/>), so I won't dive too deep here. But let me walk you through the basic anatomy of a scene file.

Our scene file has three top-level elements:

1. **Import** (<https://mapzen.com/documentation/tangram/import/>) allows us to pull in other Tangram scene files.

2. **Sources** (<https://mapzen.com/documentation/tangram/sources/>) is a required element that we use to define our data sources.
3. **Layers** (<https://mapzen.com/documentation/tangram/layers/>) is another required element that divides our data into layers that can be styled.

## Import

In this case, we use `import` to pull in the **Mapzen Refill style** (<https://mapzen.com/blog/introducing-refill-cinnabar-and-zinc-styles-for-tangram/>). (*The 6 in the url refers to the current **Refill version number** (<https://mapzen.com/documentation/cartography/styles/#refill>).*) Any other data we add to this scene will be displayed in addition to this pre-defined scene file.

And that brings us to our data...

## Sources

In honor of the 🎉 **100th birthday of the National Park Service** (<https://www.nps.gov/subjects/centennial/index.htm>) 🎉, I thought I would take a look at one of their smaller parks, **San Juan National Historic Park** (<https://www.nps.gov/sajh>), located on San Juan Island in Washington.

San Juan Island is best known for **that time when** ([https://en.wikipedia.org/wiki/Pig\\_War\\_\(1859\)](https://en.wikipedia.org/wiki/Pig_War_(1859))) the U.S. and Great Britain nearly went to war over the death of a pig.

For this exercise, we'll be using National Park Service data, **downloaded from Data.gov** (<https://catalog.data.gov/dataset/national-park-boundariesf0a4c>), which includes polygons for all of the national parks in the U.S. I went ahead and clipped just the San Juan NHP boundaries and added the data as GeoJSON to a **github gist** (<https://gist.github.com/rfriberg/684645c22f495b4a46f29fb312b6d268>), so we can link to it directly in its **raw format** ([https://gist.githubusercontent.com/rfriberg/684645c22f495b4a46f29fb312b6d268/raw/843ed38a3920ed199082636fe198ba995f5cfc04/san\\_juan\\_nhp.geojson](https://gist.githubusercontent.com/rfriberg/684645c22f495b4a46f29fb312b6d268/raw/843ed38a3920ed199082636fe198ba995f5cfc04/san_juan_nhp.geojson)).

## Layers

Once we have a data source defined, we can reference that source name when we define our layers. Here's what a basic set of layers might look like:

```
layers:
  _layer_one:
    data: { source: _source_name }
    draw: ...
  _layer_two:
    data: { source: _source_name }
    draw: ...
```

*Note: If you haven't worked with Tangram's YAML format before (or even if you have), the syntax can get a little confusing. To help differentiate between **reserved keywords** (<https://mapzen.com/documentation/tangram/yaml/#reserved-keywords>) (like `data` and `draw`) and custom element names (like `_layer_one`), I'll use an underscore before each custom name that is not reserved. No reserved words start with an underscore.*

The **draw element** (<https://mapzen.com/documentation/tangram/draw/>) is where all of our styling magic happens, and we'll focus on that in detail next time. For now, let's set up a simple style for our national park polygons.

## Styling our Data

In your scene file, add the following `draw` block:

```
draw:
  polygons:
    color: '#BCE3B4'
    order: global.sdk_order_under_roads_0
```

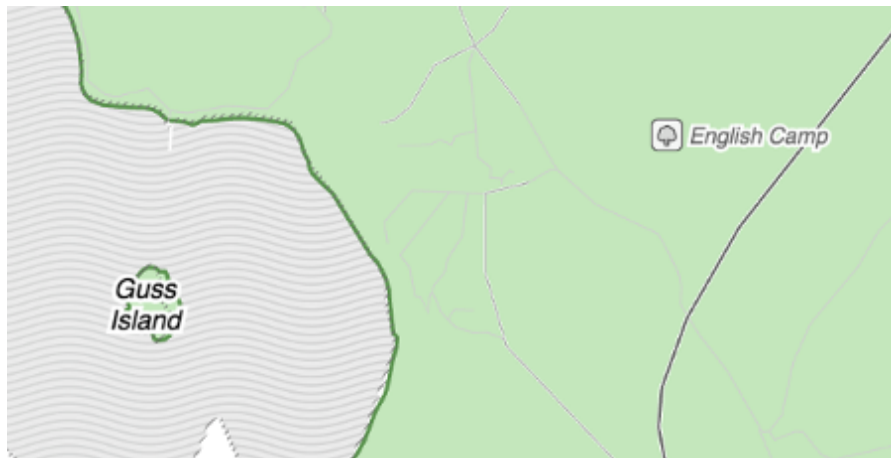
Tangram's YAML parser accepts `color` in a **wide variety of formats** (<https://mapzen.com/documentation/tangram/draw/#color>), including hex, rgb, hsl, and named colors. We'll get to that `order` parameter in just a minute.

While we're here, let's also add a border around our polygons by setting up a second draw style called `lines` :

```
lines:  
  width: 2px  
  color: '#518946'  
  order: global.sdk_order_under_roads_1
```

Reload your map and check it out.

If we zoom in...

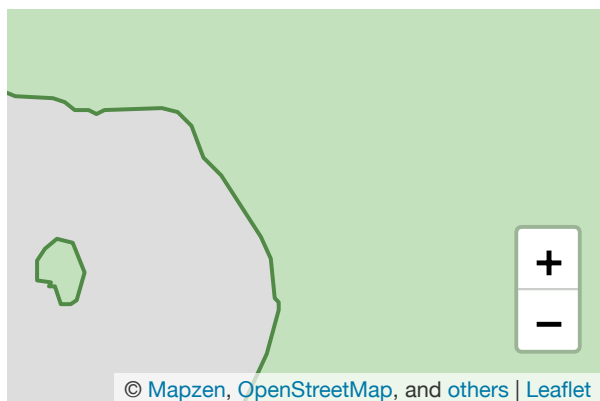


What's that you say? The labels are on top of your polygons?!

Why, you've made a map sandwich!

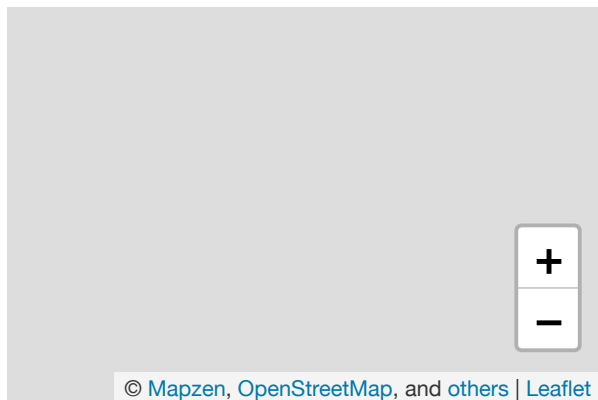
## The Map Sandwich

Let's take a look at this map alongside the Leaflet map I referenced above, so you can really see the difference:



*GeoJSON as a classic data overlay*



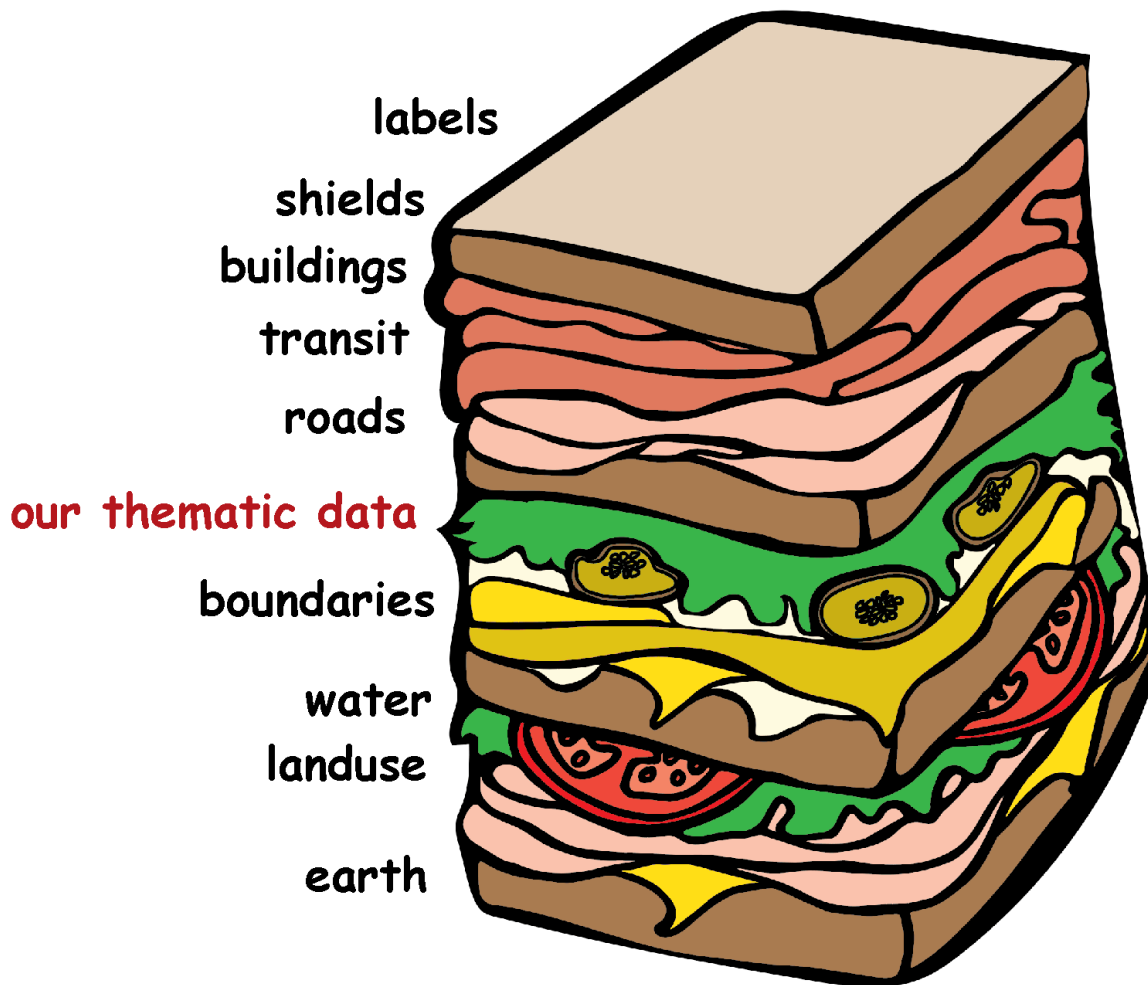


*GeoJSON via a Tangram scene file*

Pretty nice, right? Not only are labels placed on top, but we also display roads on top as well. This is especially important for providing location context to a map.

Setting up our own Tangram scene file, as we did here, gives us much more control over the placement of our layer. We can display our data in between the other layers, according to the **order** (<https://mapzen.com/documentation/tangram/draw/#order>) parameter that we set, and ensure that labels (and roads) remain on top.

In fact, we could add and reorder *lots* of layers and take our classic map sandwich into **Dagwood** ([https://en.wikipedia.org/wiki/Dagwood\\_sandwich](https://en.wikipedia.org/wiki/Dagwood_sandwich)) territory:



## A note about ordering

Tangram draws each layer according to the numeric **order** (<https://mapzen.com/documentation/tangram/draw/#order>) that is set by the scene file. Remember, we are importing an entire other scene file into our own, so layer ordering will depend on what is set in that scene file as well as in our own.

In this case, our imported Mapzen style is relying on a set of **feature ordering rules** (<https://mapzen.com/documentation/vector-tiles/layers/#feature-ordering>) specific to **Mapzen vector tiles** (<https://mapzen.com/documentation/vector-tiles/>). With so many potential layers going into a complex basemap, you might see how ordering can get complicated... *fast*.

To make ordering within a large scene file (like one of Mapzen's **basemap styles** (<https://mapzen.com/documentation/cartography/styles/>)) a little easier to swallow, our cartography team has developed a **set of global ordering variables** (<https://mapzen.com/documentation/cartography/api-reference/#data-visualization>) that

we can use in our own scene file. This is where the `global.sdk_order_under_roads_0` comes in. This variable (which happens to equate to `290` in this version of Refill) ensures our polygons are displayed under roads and labels (which have higher `order` values) and above pretty much everything else.

Let's be honest here. Ordering layers on a map can be a tricky business. Creating global variables makes this easier, but we won't stop there. We're going to continue working to take some of that trickery out of working with complex scene files. Stay tuned for more updates. And as with all of our open source projects, **discussion and contributions are welcome** (<https://github.com/tangrams/refill-style>)!

---

Next time, we'll push our style a little further and look at how we can use JavaScript functions inside of our scene file to do complex filtering and styling. (*Hello, choropleth maps...*)

But before you go, let's check out one more thing. I think you'll like it.

## Extra Credit

In your scene file, change your line width:

```
width: 2px
```

to:

```
width: [[8, 0.5px], [18, 5px]]
```

Here, we are changing the width of the line from a fixed pixel value to a dynamically changing value, using **stops** (<https://mapzen.com/documentation/tangram/yaml/#stops>). In this example, the line width will be `0.5px` at zoom 8 and below. As we zoom in, the value increases (by linear interpolation) until we hit the max. In our case, we'll top out at `5px` at zoom level 18.

By the way, this also works with width in *meters*, which is the default unit for `width`.



The full code for this exercise can be seen at

**<https://bl.ocks.org/rfriberg/639dcdbe116d4bd1898ed29c754023c3>**  
(**<https://bl.ocks.org/rfriberg/639dcdbe116d4bd1898ed29c754023c3>**).

As always, **drop us a line** (**<mailto:hello@mapzen.com>**) if you have questions or want to show off something you've made with Mapzen. We love to hear from you!

*Map sandwiches courtesy of Mapzen's own **@KatieKowalsky** (**<https://twitter.com/KatieKowalsky>**) and **@burritojustice** (**<https://twitter.com/burritojustice>**).*

~~~

Check out additional tutorials from the **Make Your Own** (**<https://mapzen.com/tag/make-your-own/>**) series:

- **One Minute Map** (**<https://mapzen.com/blog/one-minute-map/>**)
- **Map Sandwich** (**<https://mapzen.com/blog/map-sandwich/>**)
- **Filters & Functions** (**<https://mapzen.com/blog/filters-and-functions/>**)
- **Put A Label On It** (**<https://mapzen.com/blog/tangram-labels/>**)
- **Interactive Mapping with Tangram** (**<https://mapzen.com/blog/tangram-interactivity/>**)
- **Lots of Dots** (**<https://mapzen.com/blog/lots-of-dots/>**)
- **Customize Your Search** (**<https://mapzen.com/blog/customize-your-search/>**)

· 04 November 2016 ·



Rhonda Friberg

Rhonda is a javascripter who makes maps, trouble, and the occasional pie.

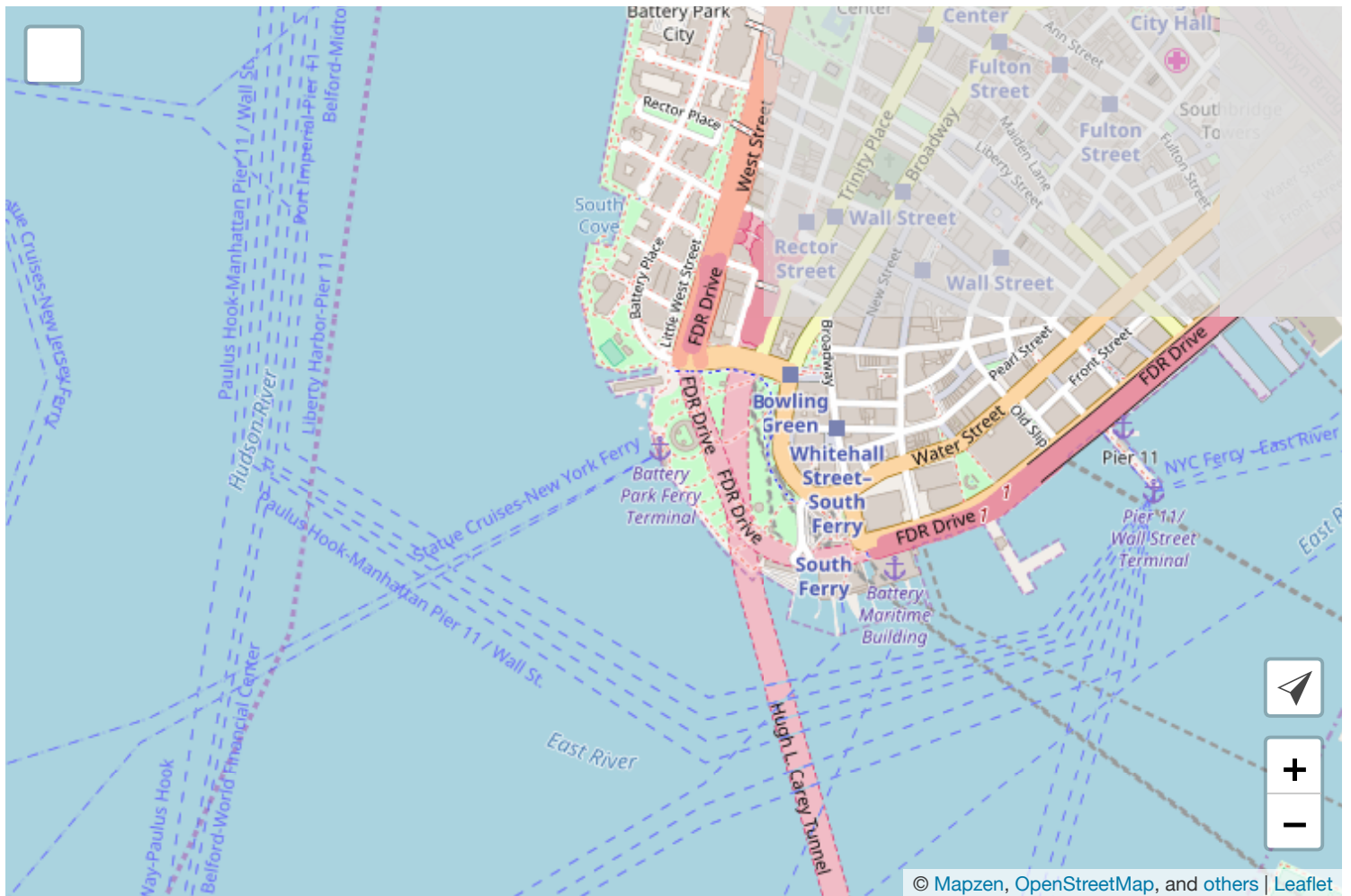
Behind the scenes, TRON2.0

tangram (/tag/tangram)

GLSL shaders are small C-like programs that modify the default rendering pipeline of your graphics card and let you manipulate every pixel on your screen at full speed. I like to think of it as the **language of light**.

One of the nicest features of the **Tangram engine** (<https://mapzen.com/products/tangram/>) is the ability to write **GLSL code** (<https://mapzen.com/documentation/tangram/Shaders-Overview/>) alongside the **YAML scene files** (<https://mapzen.com/documentation/tangram/Scene-file/>). This give Tangram's styles remarkable flexibility.

TRON2.0 was designed with the intention of pushing this power to the limits.



Everything starts with **Geraldine** (<https://twitter.com/sensescape>)'s idea of making a map where the patterns change with the zoom level, **providing an extra sense of scale** (<https://mapzen.com/blog/tron-v2-visual-scale/>). From there we started collecting visual references in a **Pinterest board** (<https://www.pinterest.com/patriciogonzv/tron-20/>).

Sketching ideas in small shaders

With some **references in mind** (<https://www.pinterest.com/patriciogonzv/tron-20/>), I started making some sketches of dynamic patterns to present to **Geraldine** (<https://twitter.com/sensescape>).

(<https://thebookofshaders.com/edit.php?log=161102162758>)

(<https://thebookofshaders.com/edit.php?log=161102162812>)

(<https://thebookofshaders.com/edit.php?log=161102162924>)

(<https://thebookofshaders.com/edit.php?log=160313020334>)

(<https://thebookofshaders.com/edit.php?log=161102163622>) (<https://thebookofshaders.com/edit.php?log=161102163943>)

(<https://thebookofshaders.com/edit.php?log=161102164303>) (<https://thebookofshaders.com/edit.php?log=161107174939>)

Hover or click on any sketch

Our next iteration consisted of making an algorithmic color palette.

```
vec3 palette(in float x) {
    return mix(vec3(0.000,1.000,1.),
               vec3(1.,0.,0.),
               vec3(smoothstep(0.0,1.048, x),
                    sin(x*2.806),
                    smoothstep(-0.512,1.072,x)))
               *(1.0-sin(-0.196+x*3.950)*0.380);
}
```

The art behind this language of light is understanding and using the expressive possibilities of numbers between 0.0 and 1.0. The palette function above, for example, takes a single value between this range and uses it to oscillate, interpolate and stretch the values between two colors, resulting in whole new spectrum of light, a spectrum that we assign programmatically to moving lines of traffic, fading buildings walls, neon highways, hillshades and shimmering oceans.

With it, we started creating the patterns that ultimately shaped the distinctive elements of TRON2.0.

(<https://thebookofshaders.com/edit.php?log=160726003844>) (<https://thebookofshaders.com/edit.php?log=160726010850>)

(<https://thebookofshaders.com/edit.php?log=161102182237>) (<https://thebookofshaders.com/edit.php?log=160621210032>)

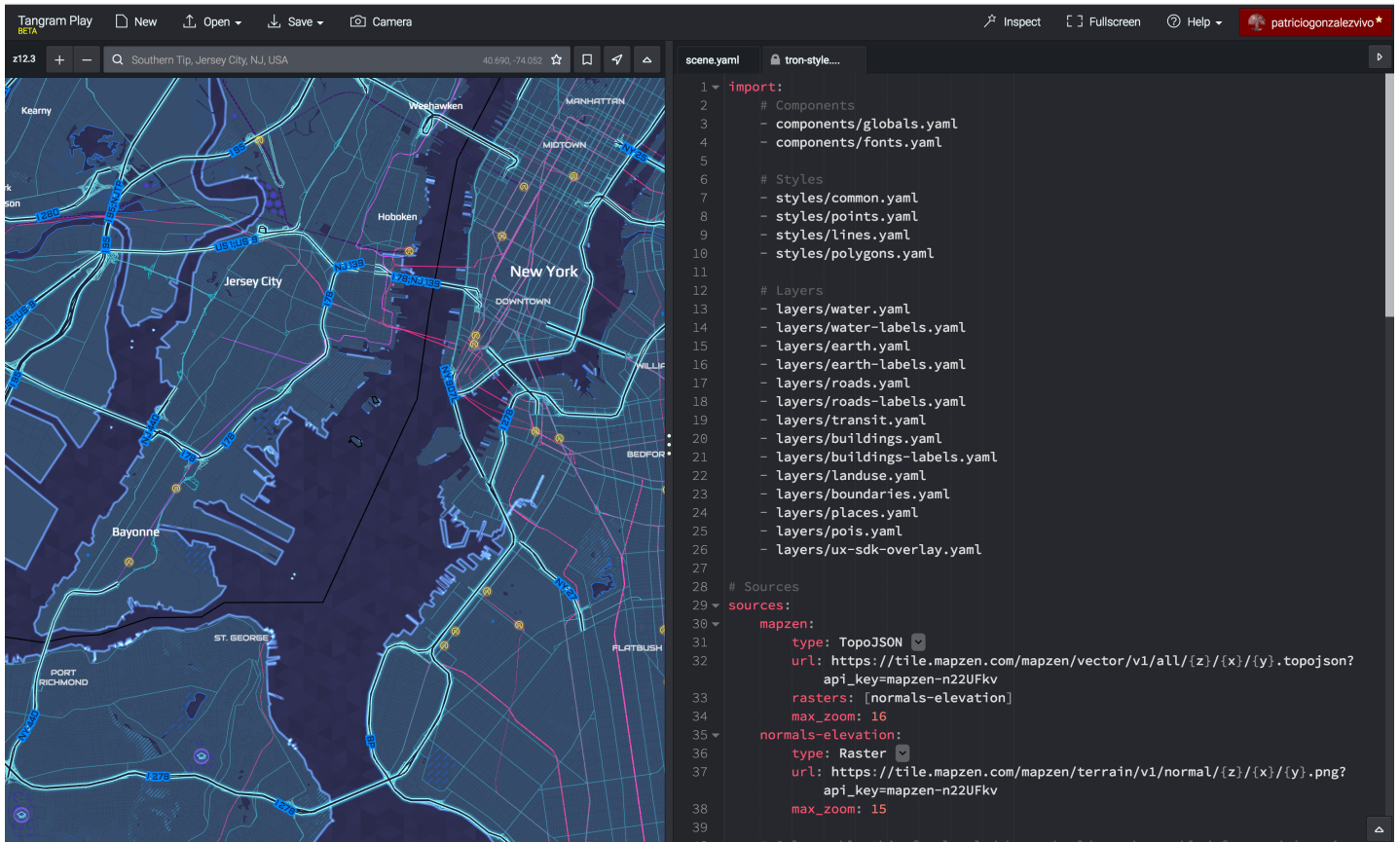
(<https://thebookofshaders.com/edit.php?log=160621170831>) (<https://thebookofshaders.com/edit.php?log=161102174317>)

(<https://thebookofshaders.com/edit.php?log=160626213924>) (<https://thebookofshaders.com/edit.php?log=161102180512>)

Hover and click on any sketch

Composing in Tangram Play

Although we used **this github repository** (<https://github.com/tangrams/tron-style/>) for the process, we also worked in our in-house web Tangram editor environment, **Tangram Play** (<https://mapzen.com/tangram/play/>).



([https://mapzen.com/tangram/play/?](https://mapzen.com/tangram/play/?scene=https%3A%2F%2Fgist.githubusercontent.com%2Fanonymous%2F14f1c7495dde62cf831427dc9be89ec9%2Fraw%2F0b25fc8a0e90115090c90928d0fefafa4423106b9b%2Fscene.yaml)

[scene=https%3A%2F%2Fgist.githubusercontent.com%2Fanonymous%2F14f1c7495dde62cf831427dc9be89ec9%2Fraw%2F0b25fc8a0e90115090c90928d0fefafa4423106b9b%2Fscene.yaml](https://gist.githubusercontent.com/anonymous/14f1c7495dde62cf831427dc9be89ec9/raw/0b25fc8a0e90115090c90928d0fefafa4423106b9b/scene.yaml))

In TRON2.0, we took advantage of one of the newest features of Tangram: the `import` system. This allows Tangram to `import` any other YAML scene file, and it is easy as...

```

import:
  - https://tangrams.github.io/tron-style/tron-style.yaml

```

...to start editing a scene.

Try it on directly in Tangram Play ([https://mapzen.com/tangram/play/?](https://mapzen.com/tangram/play/?scene=https%3A%2F%2Fgist.githubusercontent.com%2Fanonymous%2F14f1c7495dde62cf831427dc9be89ec9%2Fraw%2F0b25fc8a0e90115090c90928d0fefafa4423106b9b%2Fscene.yaml)

[scene=https%3A%2F%2Fgist.githubusercontent.com%2Fanonymous%2F14f1c7495dde62cf831427dc9be89ec9%2Fraw%2F0b25fc8a0e90115090c90928d0fefafa4423106b9b%2Fscene.yaml](https://gist.githubusercontent.com/anonymous/14f1c7495dde62cf831427dc9be89ec9/raw/0b25fc8a0e90115090c90928d0fefafa4423106b9b/scene.yaml))

We used `import` to divide the scene into different folders and files, making TRON2.0 **easier to read and edit**.

Name	^	Date Modified	Size	Kind
▶ blocks		Today, 3:20 PM	--	Folder
▼ components		Today, 3:19 PM	--	Folder
▶ fonts		Today, 3:19 PM	--	Folder
fonts.yaml		Today, 3:19 PM	930 bytes	YAML
globals.yaml		Today, 3:19 PM	10 KB	YAML
▶ images		Today, 3:19 PM	--	Folder
index.html		Today, 3:19 PM	24 KB	HTML
▼ layers		Today, 3:19 PM	--	Folder
boundaries.yaml		Today, 3:19 PM	2 KB	YAML
buildings-labels.yaml		Today, 3:19 PM	3 KB	YAML
buildings.yaml		Today, 3:19 PM	4 KB	YAML
earth-labels.yaml		Today, 3:19 PM	532 bytes	YAML
earth.yaml		Today, 3:19 PM	913 bytes	YAML
landuse.yaml		Today, 3:19 PM	14 KB	YAML
places.yaml		Today, 3:19 PM	58 KB	YAML
pois.yaml		Today, 3:19 PM	61 KB	YAML
roads-labels.yaml		Today, 3:19 PM	6 KB	YAML
roads.yaml		Today, 3:19 PM	29 KB	YAML
transit.yaml		Today, 3:19 PM	3 KB	YAML
ux-sdk-overlay.yaml		Today, 3:19 PM	6 KB	YAML
water-labels.yaml		Today, 3:19 PM	8 KB	YAML
water.yaml		Today, 3:19 PM	8 KB	YAML
▶ lib		Today, 3:19 PM	--	Folder
LICENSE		Today, 3:19 PM	1 KB	TextEd...ument
README.md		Today, 3:19 PM	8 KB	Markdown
▼ styles		Today, 3:19 PM	--	Folder
common.yaml		Today, 3:19 PM	662 bytes	YAML
lines.yaml		Today, 3:19 PM	3 KB	YAML
points.yaml		Today, 3:19 PM	14 KB	YAML
polygons.yaml		Today, 3:19 PM	4 KB	YAML
tron-style.yaml		Today, 3:19 PM	4 KB	YAML

Here's how it ends up looking:

- **tron-style.yaml** (<https://github.com/tangrams/tron-style/blob/gh-pages/tron-style.yaml>) is the main `.yaml` file that mixes and holds it all together. There you will find the definition of the `sources`, `cameras` and `scene:background:color`, together with the `imports` that glow and connections to the `layers`, `styles` and `components` resources.
- the **layers/** folder (<https://github.com/tangrams/tron-style/tree/gh-pages/layers>) holds the sets of rules that filter the data (coming from the `sources`) into different **style rules**. **layers** (<https://mapzen.com/documentation/tangram/Filters-Overview/>) tell Tangram how to treat each component on the map, whether they are `text`, `points`, `lines` or `polygons`, with `size` and `color`. In this folder you will find different `.yaml` scene files that carefully make sense of the data and display it in a cartographically intuitive way. This meticulous work was made by **Geraldine Sarmiento** (<https://twitter.com/sensescape>) and **Nathaniel V. Kelso** (<https://twitter.com/kelsosCorner>). At the end of each layer file you will also find a `styles` section where the **custom shaders** are defined and crafted by **me (Patricio Gonzalez Vivo)**. (<https://twitter.com/patriciogv>) Shaders are not a simple thing, but at

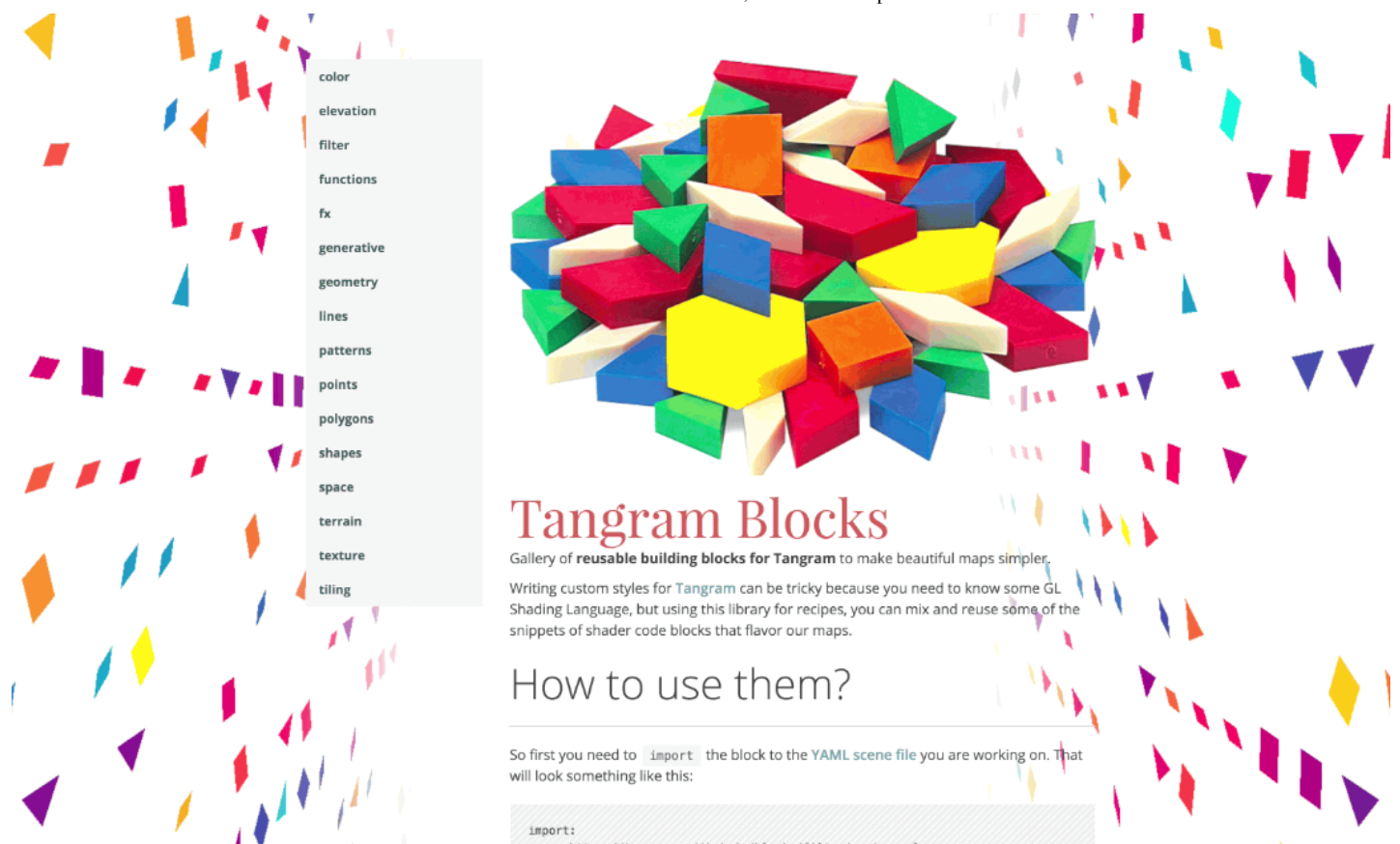
Mapzen (<https://mapzen.com>) we are trying to make them more approachable. Tangram styles can be mixed, which reduces the complexity and size of the shaders by reusing them as much as possible.

- the **styles/ folder (<https://github.com/tangrams/tron-style/tree/gh-pages/styles>)** contains all the styles that are shared across layers. It is definitely a more **abstract** layer. This folder sorts the styles by their base geometry (points , lines or polygons). Those styles common to all the other geometries can be found in `common.yaml` . You will note that most **shader styles (<https://mapzen.com/documentation/tangram/Shaders-Overview/>)** “extend” from **blocks (<https://tangrams.github.io/blocks/>)**.
- the **components/ folder (<https://github.com/tangrams/tron-style/tree/gh-pages/components>)** holds some of the global resources used in the scene, like **fonts (`components/fonts.yaml`)** and **images/ (`components/images`)** along with the **`globals.yaml` (`components/globals.yaml`)**. This particular file is very interesting because holds `globals` variables and JS functions that control the map. We can use these `globals` to turn on or off the animations or to change the language on the map.

Reusing the building blocks

“When coding, don’t repeat yourself.”

Thanks to the new `import` feature, I was able to turn shader recipes into something that anybody could use in Tangram styles. As part of the effort to make GLSL Shaders more approachable in Tangram, I started building **Tangram Blocks (<http://tangrams.github.io/blocks/>)**, a toolbox/library of shader snippets that can be mixed together. A lot of work on TRON2.0 involved making these blocks reusable in a production context. Each block self-documents itself, providing not only a description and examples but also helpful information like the range of the expected variables.



(<https://tangrams.github.io/blocks/>)

This is a list of some of the new patterns introduced by TRON2.0:

- **Polygons diagonal stripes** (<https://tangrams.github.io/blocks/#polygons-diagonal-stripes>)
- **Polygons dots** (<https://tangrams.github.io/blocks/#polygons-diagonal-dots>)
- **Polygons shimmering** (<https://tangrams.github.io/blocks/#polygons-shimmering>)
- **Lines glow** (<https://tangrams.github.io/blocks/#lines-glow>)
- **Lines dots glow** (<https://tangrams.github.io/blocks/#lines-dots-glow>)
- **Lines data stream** (<https://tangrams.github.io/blocks/#lines-datastream>) for the car animation
- **Elevation stripes hillshading** (<https://tangrams.github.io/blocks/#elevation-stripes>)
- **Interpolation zoom function** (<https://tangrams.github.io/blocks/#functions-zoom>)

Bundling things up

By the time we finished, we had a beautiful animated map. We also had a lot of related YAML scene files. Each one of them needs to be loaded before we can start rendering – this would mean a lot of HTTP requests that would affect performance.

To solve that I worked up a small Python script that lets you bundle a scene file and its dependencies and components all together into one .zip file, called **Tangram Bundler** (<https://github.com/tangrams/bundler>).

Installing it is simple:

```
pip install tangram_bundler  
tangram-bundle scene.yaml
```

You use it by referring to the main YAML scene file (the one that calls all the others).



(<https://github.com/tangrams/bundler>)

TRON2.0 was a trip of innovative concepts and ideas that we took with **Geraldine** (<https://twitter.com/sensescape>). It was as much fun as it was aesthetically and technically challenging. Let us know how you use it!

· 08 November 2016 ·



Patricio Gonzalez Vivo

Patricio is an artist and graphic engineer who speaks the language of light.



Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

© 2017 Mapzen

How to make your code backwards (and compatible)

tangram (/tag/tangram)

In Lewis Carroll's *Through the Looking Glass*, Alice finds a mysterious book that's "all in some language [she doesn't] know". It begins with

YKCOWREBBAJ

*sevot yhtils eht dna,gillirb sawT'
ebaw eht ni elbmig dna eryg diD
,sevogorob eht erew ysmim lIA
.ebargtuo shtar emom eht dnA*

Tangram.min.js (<https://github.com/tangrams/tangram/blob/master/dist/tangram.min.js>)

looked somewhat similar between versions 0.10.0 and 0.10.5. A large part of our code was *backwards*. Like the poem of Jabberwocky, you need a mirror to read it.

```
=r,[ ]=t rav)rof}(e)a noitcnuf{(e)tset.n nruter;("[+r+"]*[+t+]^")pxEgeR
rav}{++n;htgnel.e>n;0=n,[ ]
{(a)hsup.t;(( )pop.r)hsup.t(;0<htgnel.r;)rof}esle;(a)hsup.r(o)fi;(a)i=o,[n]e=a
")tilps.e=t rav}(e)o noitcnuf{t nruter;(( )pop.r)hsup.t(;0<htgnel.r;)rof{
rav;({0!:eulav},"eludoMse__",r)ytreporPenifed.tcejbo{(t)a nruter;("
-lebab")e=u,(s)n=l,("yarrAoTdecils/srepleh/emitnur-lebab")e=s
-lebab")e=f,(u)n=c,("kcehCllaCssalc/srepleh/emitnur
/..")e=m,(d)n=p,("slitu/slitu/./..")e=d,(f)n=h,("ssalCetaerc/srepleh/emitnur
es_gubed/slitu/./..")e=y,(g)n=v,("reganam_tnof/.")e=g,(m)n=_,("erutxet/lg/..
( )e noitcnuf}{ )noitcnuf=x,(y)n=b,("sgnitt
s.savnac.siht,("savnac")tnemelEetaerc.tnemucod=savnac.siht,(e,siht)tluafe.d.c}
,("d2")txetnoCteg.savnac.siht=txetnoc.siht,"tnerapsnart"=roloCdnorgkcab.elyt
,e)tluafe.d.h nruter{4=reffub_txet_latnoziroh.siht,8=reffub_txet_lacitrev.siht
(t,e)noitcnuf:eulav,"eziser":yek}
,{(t,e,0,0)tceRracl.txetnoc.siht,t=thgieh.savnac.siht,e=htdiw.savnac.siht}
rav}(e)noitcnuf:eulav,"tnoFtes":yek}
;a=ezis_xp.siht;ezis_xp.e=a,htdiw_ekorts.e=i,ekorts.e=n,llif.e=r,ssc_tnof.e=t
&&0<i&&n;oitar_lexip_ecived.tluafe.d.p=s,txetnoc.siht=o rav
,{2=timilRetim.o,t=tnof.o,r=elytSllif.o,(s*i=htdiWenil.o,n=elytSekorts.o)
rav}(t)noitcnuf:eulav,"seziStxet":yek}
nruter;oitar_lexip_ecived.tluafe.d.p=n,siht=r
(t ni i rav)rof}{ )noitcnuf)neht.( )stnoFdaol.tluafe.d.v
```

sj.nim.margnaT

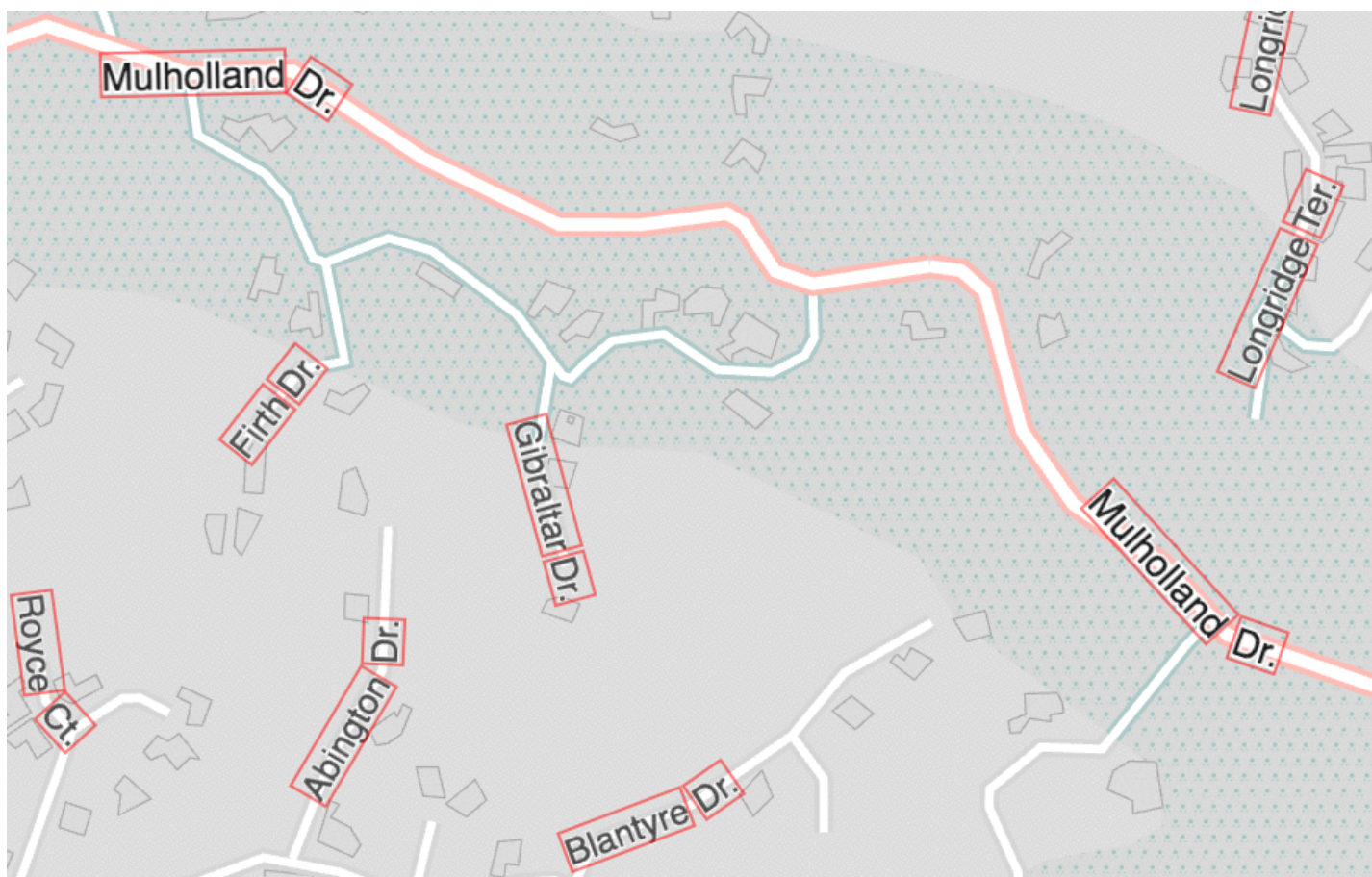
And yet, everything still worked as expected! The code only appears to go left, but its logic is all right. You can verify it for yourself by executing this simpler JavaScript example:

```
';() ( {;('ssalg gnikool eht hguorht')gol.elosnoc }() noitcnuf);'
```

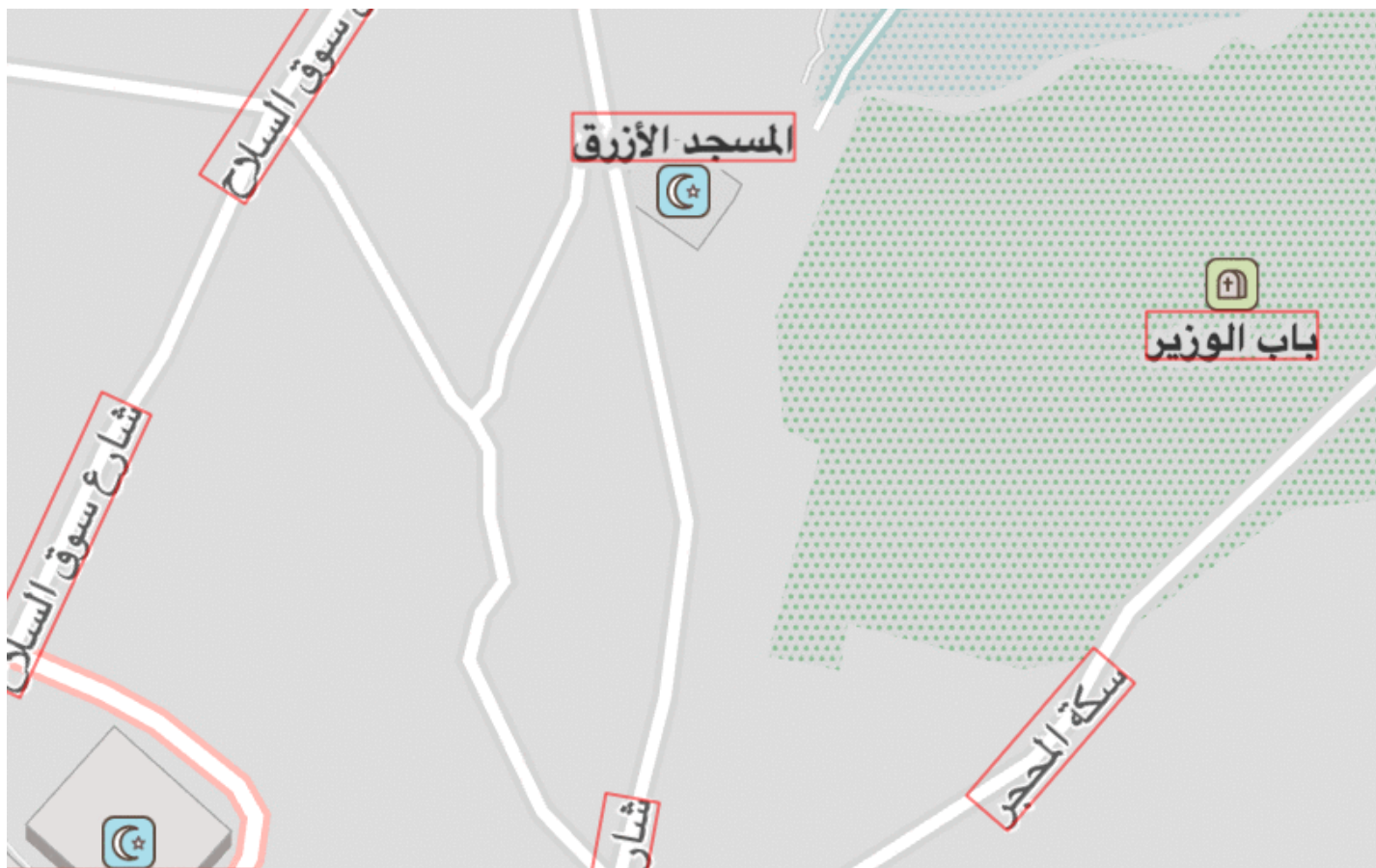
Here's the story of how this happened, and how you, too, can make your code backwards (and compatible).

Maps and Labels

First, some context. Our maps render text in hundreds of languages. One place we do this is to label roads, and we've been recently experimenting with curving text along roads. This comes down to breaking up a label's text into pieces, and reassembling them along a curve.



Taking Apart and Putting Back Together



Click here (<https://tangrams.github.io/bubble-wrap/#14.77084/32.6791/35.2457>) to see these labels adjust live in Tangram.

Escaping Unicode

When unicode is interpreted, it is immediately escaped, turning it into the text it represents. This way you can type `\u00A9` and get the copyright symbol ©. When our source code was bundled by browserify, our unicode was escaped and the code above became

```
function isRTL(string) {  
    var weakChars = '\\u0000-@[-`{-ix÷'-← -□--\\u2029 -□',  
        rtlChars = '،'ـٔٱللهآأخزألجسبءَئِىٰ۞؎  
        rtlDirCheck = new RegExp('^[' + weakChars + ']*([' + rtlChars + ']')');  
  
    return rtlDirCheck.test(string);  
}
```

Surprisingly, this code still works! `isRTL("hello")` is still `true`. Some may even argue this version is just as cryptic than the original. Nonetheless, our codebase has some new and colorful characters. You can see the Hebrew Hiriq (׳), the symbol ☐

(<http://www.fileformat.info/info/unicode/char/07ff/index.htm>) (which is the “not a valid

character" character), and the Arabic **Basmala** (<https://en.wikipedia.org/wiki/Basmala>) (بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ). The Basmala is an Arabic phrase, drawn as a single unicode character, which translates to "In the name of God, the Most Gracious, the Most Merciful". A religious mantra snuck into our source code!

Now we can explain why our minified source code was backwards. One of the unicode characters in our regular expression above, `\u202E`, is the **right-to-left override** (<http://www.fileformat.info/info/unicode/char/202e/index.htm>) character, which forces all subsequent text to render right-to-left until there is either a newline character, or a left-to-right override character to stop it. The RTL override character then acted on a large chunk of our own source code! Before minification, it only acted on the line on which it appeared. After minification this effect became exaggerated because the thousands of lines of our codebase collapsed to about 20, and a large part of one of those lines was rendered backwards.

You, too, can write code backwards

Humpty Dumpty took the book, and looked at it carefully. "That seems to be done right —" he began.

"You're holding it upside down!" Alice interrupted.

We thought this was so cool that we made an **npm script** (<https://github.com/dmvaldman/elba>) for it. You can install it with

```
npm install -g elba
```

Making the code in `myFile.js` backwards is as simple as

```
elba myFile.js > eliFym.js
```

For example, **here's**

(<https://gist.githubusercontent.com/dmvaldman/8238a65a460ffea046888c1872db66f5/raw/93524334384bda6c0a668a717d785ffad4034fb2/erocsrednu.js>) `elba` applied to the popular **underscore.js** (<https://github.com/jashkenas/underscore>) library. And yes, it works! 🤖

*Preview image is an original illustrations from Alice's Adventures in Wonderland, drawn by John Tenniel, via **alice-in-wonderland.net** (<http://www.alice-in-wonderland.net/resources/pictures/alices-adventures-in-wonderland/>)*

· 09 November 2016 ·



David Valdman

Working on Tangram. Henceforth, it is the map that precedes the territory.

© 2017 Mapzen

Filters and Functions

mapzen-js (/tag/mapzen-js) **demo** (/tag/demo) **tutorial** (/tag/tutorial)

tangram (/tag/tangram) **make-your-own** (/tag/make-your-own)

Welcome back! This is our third installment of a new series that we are calling **Make Your Own** (<https://mapzen.com/tag/make-your-own/>). If you've missed the last two, I recommended giving those a read, as they lay the groundwork for getting started with **mapzen.js** (<https://mapzen.com/documentation/mapzen-js/>) and the **Tangram** (<https://mapzen.com/documentation/tangram/>) scene file:

- **One Minute Map** (<https://mapzen.com/blog/one-minute-map/>)
- **Map Sandwich** (<https://mapzen.com/blog/map-sandwich/>)

Last time, I showed you how to make a **map sandwich** (<https://mapzen.com/blog/map-sandwich/>) by inserting your data layer *in between* the other map layers displayed on the map. This resulted in labels on top (and, if you are anything like me, a whole lot of celebratory dancing 🎉).

This week, we'll push our map sandwich even further by setting up filters and using JavaScript to create a custom style for our data.

Let's jump right in, shall we?

*Note: The following instructions assume you have a text editor and access to a web server. If you need either, read our documentation on **setting up a bare bones development environment** (<https://mapzen.com/documentation/guides/>) or use the gist → *blocks workflow* described in **One Minute Map** (<https://mapzen.com/blog/one-minute-map/>).*

We'll start with the same two files we created last time, with some minor changes to our national park polygons.

index.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>San Juan Island Geology</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="https://mapzen.com/js/mapzen.css">
    <script src="https://mapzen.com/js/mapzen.min.js"></script>
    <style>
      #map {
        height: 100%;
        width: 100%;
        position: absolute;
      }
      html,body{margin: 0; padding: 0}
    </style>
  </head>
  <body>
    <div id="map"></div>
    <script>
      // Mapzen API key (replace key with your own)
      // To generate your own key, go to https://mapzen.com/developers/
      L.Mapzen.apiKey = 'mapzen-JA21Wes';

      var san_juan_island = [48.5326, -123.0879];

      var map = L.Mapzen.map('map', {
        center: san_juan_island,
        zoom: 12,
        scene: 'scene.yaml',
      });

      // Move zoom control to the top right corner of the map
      map.zoomControl.setPosition('topright');

      // Mapzen Search box (replace key with your own)
      // To generate your own key, go to https://mapzen.com/developers/
      var geocoder = L.Mapzen.geocoder('mapzen-JA21Wes');
      geocoder.addTo(map);

    </script>
  </body>
</html>
```

UPDATE March 1, 2017: A Mapzen developer API key is now required for mapzen.js. We've updated the Make Your Own series to include a demo key. Generate your own free API key at <https://mapzen.com/developers/> (<https://mapzen.com/developers/>).

scene.yaml:

```
import: https://mapzen.com/carto/refill-style/6/refill-style.zip

sources:
  _nps_boundary:
    type: GeoJSON
    url: https://gist.githubusercontent.com/rfriberg/684645c22f495b4a46f29fb312b6d268,

layers:
  _national_park:
    data: { source: _nps_boundary }
    draw:
      lines:
        visible: false
        width: [[8, 0.5px], [18, 5px]]
        color: '#518946'
        order: global.sdk_order_under_roads_1
```

Notice that we removed the polygons, but kept the boundary lines. Though, if you view the map now, the lines won't display because we set **visible: false** (<https://mapzen.com/documentation/tangram/draw/#visible>). We'll come back to those later.

As before, we are going to focus on San Juan Island in Washington. While San Juan is best known for **that time when** ([https://en.wikipedia.org/wiki/Pig_War_\(1859\)](https://en.wikipedia.org/wiki/Pig_War_(1859))) the U.S. and Great Britain nearly went to war over the death of a pig, we're going to take a look at the island's geology. (I'll try to keep the rock puns to a minimum.)

The geology dataset we will be using began its journey as a shapefile of polygons representing geologic units for San Juan Island, downloaded from **Data.gov** (<https://catalog.data.gov/dataset/digital-geologic-map-of-san-juan-island-national-historical-park-and-vicinity-washington-nps-gr>). Currently, **Tangram supports four data types** (<https://mapzen.com/documentation/tangram/sources/#type>): GeoJSON, TopoJSON, Mapbox Vector Tiles, and Raster. So, I loaded the shapefile into my trusty **QGIS** (<http://www.qgis.org/en/site/>), exported as a GeoJSON file, and uploaded the GeoJSON to <https://gist.github.com> (<https://gist.github.com/rfriberg/3c09fe3afd642224da7cd70aff1c1e70>).

Let's update our scene.yaml file to add the gist's **raw GeoJSON**

(https://gist.githubusercontent.com/rfriberg/3c09fe3afd642224da7cd70aff1c1e70/raw/1f1df59f4cb4e82d7ea23452c789bc99c299a5cb/san_juan_nhp_geology.geojson) as a new data source:

```
_nps_geology:
  type: GeoJSON
  url: https://gist.githubusercontent.com/rfriberg/3c09fe3afd642224da7cd70aff1c1e70,
```

We'll then add a new layer called `_geology` that pulls in the new data source and sets up a simple polygon style:

```
_geology:
  data: { source: _nps_geology }
  draw:
    polygons:
      order: global.sdk_order_under_roads_0
      color: red
```

Your scene file and map should now look like this:

scene.yaml

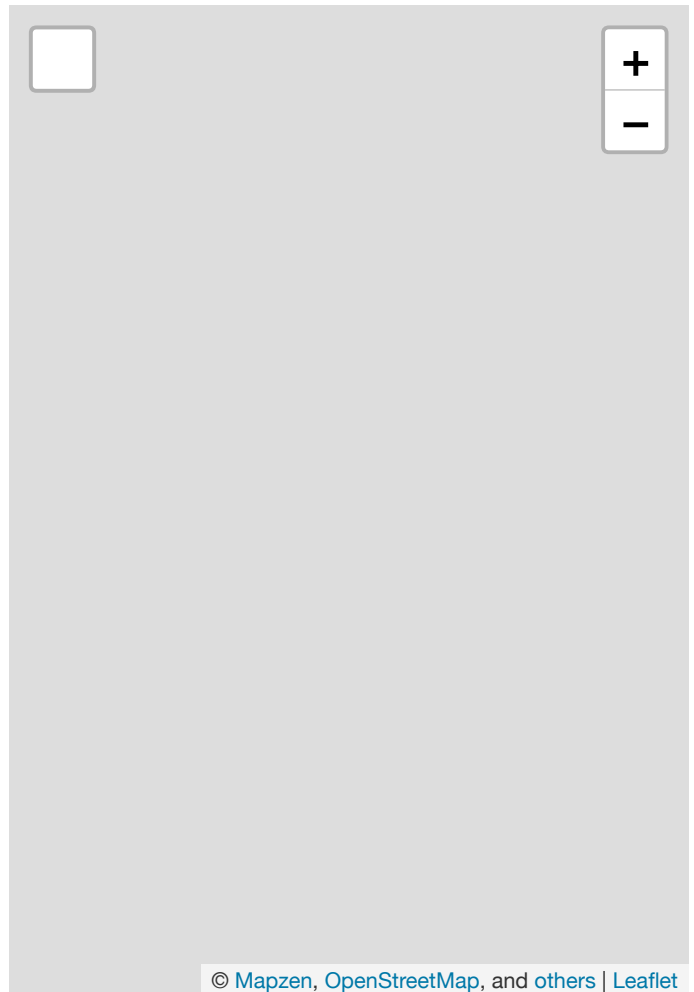
```
import: https://mapzen.com/cartto/refill-:

sources:
  _nps_boundary:
    type: GeoJSON
    url: https://gist.githubusercont

  _nps_geology:
    type: GeoJSON
    url: https://gist.githubusercont

layers:
  _national_park:
    data: { source: _nps_boundary }
    draw:
      lines:
        visible: false
        width: [[8, 0.5px], [18,
        color: '#518946'
        order: global.sdk_order_

  _geology:
```



It's not pretty, but we'll fix that.

Filtering

The first thing I want to do, is to remove any polygons representing “water”.

If you take a look at our **GeoJSON file**

(https://gist.githubusercontent.com/rfriberg/3c09fe3afd642224da7cd70aff1c1e70/raw/1f1df59f4cb4e82d7ea23452c789bc99c299a5cb/san_juan_nhp_geology.geojson), you'll see that each feature comes with a set of properties:

```
{
  "type": "Feature",
  "properties": {
    "FUID": 1,
    "GLG_SYM": "KJmm(c)",
    "SRC_SYM": "KJm(c)",
    "SORT_NO": 13.0,
    "NOTES": "NA",
    "GMAP_ID": 74832,
    "HELP_ID": "KJmm(c)",
    "SHAPE_Leng": 137.95199379300001,
    "SHAPE_Area": 954.47301117400002
  },
  "geometry": ...
}
```

The `GLG_SYM` property contains the geologic symbol for each rock unit; in this case, that `KJmm(c)` refers to a marine metasedimentary layer from the Cretaceous-Jurassic (think **Plesiosaurs** (<https://en.wikipedia.org/wiki/Plesiosauria>)). Looking closely at the GeoJSON, it appears that “water” has been lumped in here, too. So now we know that we can use the `GLG_SYM` property to filter out “water”.

There are several **filter functions** (<https://mapzen.com/documentation/tangram/Filters-Overview/#filter-functions>) available to help filter features in our scene file. One easy way is with a `not` function. Let’s add that to our scene, just below the `data` block:

```
data: { source: _nps_geology }
filter:
  not: { GLG_SYM: water }
```

Super simple, right? Tangram’s YAML parser is smart enough to know that `GLG_SYM` is the name of a **feature property** (<https://mapzen.com/documentation/tangram/Filters-Overview/#feature-properties>) and `water` is one of the values stored on that property.

Let’s say we also want to filter out the data past a certain zoom level. We can use the filter function `all` to filter out the water *and* limit the remaining data to zoom level 10 and above.

```
filter:
  all:
    - { $zoom: { min: 10 } }
    - not: { GLG_SYM: water }
```

This will keep our map from trying to render data at a zoom level where that data wouldn't be useful anyway. Go ahead and zoom out. You should see the red polygons disappear after z10.

By the way, that `$zoom` keyword we used above is one of Tangram's **reserved keywords** (<https://mapzen.com/documentation/tangram/yaml/#reserved-keywords>). Tangram's YAML parser has two particular keywords (`$zoom` and `$geometry`) that are extremely useful for defining filters. In addition to `$zoom` , we could also filter our data by `$geometry` , which allows us to limit our data to only `point` , `line` , or `polygon` features. Very helpful if your dataset contains multiple geometry types. (*I'm looking at you OSM* (<http://www.openstreetmap.org/>).)

Custom styles

We now have filtered data. Let's work on a style for these polygons.

In last week's post, we used `polygons` and `lines` to draw, well... polygons and lines. What those really refer to are built-in **draw styles** (<https://mapzen.com/documentation/tangram/Styles-Overview/#draw-styles>). There are five of these built-in draw styles in Tangram: `polygons` , `lines` , `points` , `raster` , and `text` . Each of these displays data in a different way, and each can be extended to get more control over the style of our data. More control means doing things like adding `transparency` or `outlines` or `anim`.

But let's not get too far ahead of ourselves. We'll start with adding some transparency.

In our scene file, we are using the built-in `polygons` draw style to display our `_geology` layer:

```
polygons:
  order: global.sdk_order_under_roads_0
  color: red
```


Let's try changing that to an rgba value, which includes a value for transparency:

```
color: rgba(255, 0, 0, 0.35)
```

Reload the map.

Notice anything? The color was de-saturated, but there's no transparency. I'll zoom into Roche Harbor Lake so you can see what I mean:



Our original "red" polygons compared to the `polygons` style with an rgba value.

Notice that the waves of the lake polygon are still blocked by our de-saturated 'red'.

That's because the default **blend mode**

(<https://mapzen.com/documentation/tangram/styles/#blend>) of the `polygons` draw style is `opaque`. The **blend mode** (<https://mapzen.com/documentation/tangram/styles/#blend>) defines the way in which color interacts with the underlying layers. Some of the modes, like `add` and `multiply` work similarly to Photoshop filters. In our case, `opaque` is simply *opaque*, and it obscures all of the underlying features. To make it semi-transparent, we need to set a blend mode of `inlay` or `overlay`.

Let's do this by creating a **styles block**

(<https://mapzen.com/documentation/tangram/styles/>) near the top of our scene file with a single custom style we'll call `_alpha_polygons` :

```
import: https://mapzen.com/carto/refill-style/6/refill-style.zip

styles:
  _alpha_polygons:
    base: polygons
    blend: overlay
```

This style extends the `polygons` style we've already worked with, but changes the blend mode from the default (`opaque`) to `overlay` .

Next, we'll update our layer with the new style name, `_alpha_polygons` :

```
draw:
  _alpha_polygons:
    order: global.sdk_order_under_roads_0
    color: rgba(255, 0, 0, 0.35)
```

And reload the map. Much better! The color is still saturated, but now we can (partially) see the waves of the underlying lake polygon.



Our original “red” polygons compared to a custom `_alpha_polygon` style with an `rgba` value.

Great. Now let’s do something a little more interesting than plain red polygons.

You know, before your patience *erodes* any further.

JavaScript Functions

Did you know that Tangram’s YAML parser accepts **JavaScript functions** (<https://mapzen.com/documentation/tangram/yaml/#function>)? I didn’t. Granted, I’m new here (https://twitter.com/rhonda_friberg/status/786952139801145344), but this was a **very exciting** realization for me.

We are already filtering out water by using the `not` filter function. Let’s try replacing that `not` filter with an equivalent JavaScript function:

```
filter:
  all:
    - { $zoom: { min: 10 } }
    - function() { return feature.GLG_SYM != 'water' }
```

As Tangram parses the YAML file, it will notice the `function` keyword and treat the text that follows as a JavaScript function. We can even add comments to our function in JavaScript's own comment syntax:

```
filter:
  all:
    - { $zoom: { min: 10 } }
    - |
      function() {
        // Filter out water
        return feature.GLG_SYM != 'water';
      }
```

The pipe (`|`) and newline preceding the function lets the parser know that a **multi-line string** (<https://mapzen.com/documentation/tangram/yaml/#multi-line-strings>) is coming up. That way, we can write our functions so that they still look like code.

Gneiss!

(I am so sorry.)

Moving on...

Let's tackle our polygon style, next. I want a different color for each geologic unit on my map, to more or less align with this **suggested range of colors** (http://pubs.usgs.gov/tm/2006/11A02/FGDCgeostdTM11A2web_Sec33.pdf) from the FGDC.

Lucky for us, Tangram passes our GeoJSON **feature properties** (<https://mapzen.com/documentation/tangram/Filters-Overview/#feature-properties>) to all JavaScript functions under the `feature` keyword.

So instead of passing in a fixed color, we can use JavaScript to return a color based on our `GLG_SYM` property:

```

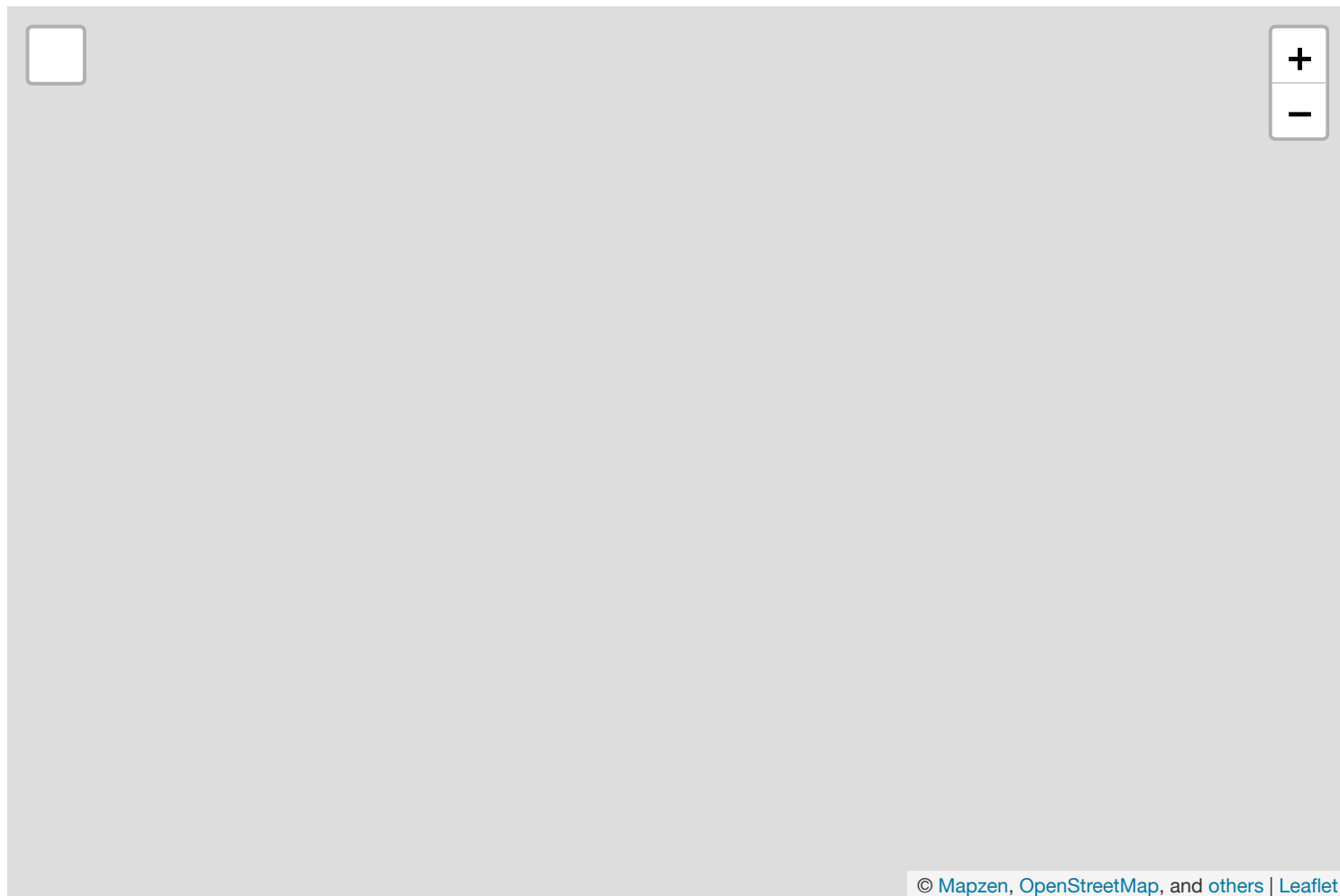
_alpha_polygons:
  order: global.sdk_order_under_roads_0
  color: |
    function() {
      var category = feature.GLG_SYM;
      var color = category == 'Qa'      ? '#FFF79A' :
                    category == 'Qb'      ? '#FFF46E' :
                    category == 'Qd'      ? '#fff377' :
                    category == 'Qf'      ? '#dddddd' :
                    category == 'Qp'      ? '#EAC88D' :
                    category == 'Qgdm'    ? '#FCBB62' :
                    category == 'Qgdm(es)' ? '#FEE9BB' :
                    category == 'Qgdm(e)' ? '#E8A121' :
                    category == 'Qgom(e)' ? '#EAB564' :
                    category == 'Qgom'    ? '#FECE7A' :
                    category == 'Qgd'     ? '#FEDDA3' :
                    category == 'Qgt'     ? '#FCBB62' :
                    category == 'KJmm(c)' ? '#86C879' :
                    category == 'KJm(ll)' ? '#9FD08A' :
                    category == 'JTRmc(o)' ? '#27BB9D' :
                    category == 'TRn'     ? '#ED028C' :
                    category == 'TRPMv'   ? '#F172AC' :
                    category == 'TRPv'    ? '#F499C2' :
                    category == 'PDmt'    ? '#40C7F4' :
                    category == 'pPsh'    ? '#9BA5BE' :
                    category == 'pDi'     ? '#848FC7' :
                    category == 'pDit(t)' ? '#B28ABF' :
                    '#000';

      return color;
    }

```

Ooh, look at that nice spacing...

Reload the map and check it out.



Look at that, we made a thematic map! This is just a simple categorization, but I bet you can already see how we can use this method for choropleths, proportional symbols, value-by-alpha, and other types of thematic maps.

Well, this is fine, but you know what would make this map even better?

Hillshading!

Whaaaaa?!

Yup. Let's do this.

Instead of Refill, let's import Mapzen's **Walkabout style** (<https://mapzen.com/documentation/cartography/styles/#walkabout>), which comes with a nice built-in hillshade:

```
import: https://mapzen.com/carto/walkabout-style/3/walkabout-style.zip
```

Next, we'll update our custom `_alpha_polygons` style to use the `multiply` blend mode:

```
_alpha_polygons:  
  base: polygons  
  blend: multiply
```

The `multiply` blend mode multiplies the color of our layer with the underlying layer colors, resulting in a darker color on screen. In this case, I want our geology polygon colors to multiply with the underlying hillshading provided by **Walkabout** (<https://mapzen.com/documentation/cartography/styles/#walkabout>). This will add some very nice texture to our final product.

At this point, our roads are still on top of our polygons. That's *ok* but I'd like them to blend in a bit better with our map. So, while we're making updates, let's bump up the order of our `_geology` layer from `global.sdk_order_under_roads_0` (the **basic underlay** (<https://mapzen.com/documentation/cartography/api-reference/#basic-underlay>)) to `global.sdk_order_over_everything_but_text_0` (the **classic overlay** (<https://mapzen.com/documentation/cartography/api-reference/#overlay>)). If you recall from last time, these `globals` are provided by Mapzen's basemap styles to make it easier for ordering your own data on a map.

Read more about ordering here (<https://mapzen.com/documentation/cartography/api-reference/#data-visualization>). *(And don't worry. Our labels will still be on top!)*

Your scene file should now look something like this:


```

import: https://mapzen.com/carto/walkabout-style/3/walkabout-style.zip

styles:
  _alpha_polygons:
    base: polygons
    blend: multiply

sources:
  _nps_boundary:
    type: GeoJSON
    url: https://gist.githubusercontent.com/rfriberg/684645c22f495b4a46f29fb312b6d268,

  _nps_geology:
    type: GeoJSON
    url: https://gist.githubusercontent.com/rfriberg/3c09fe3afd642224da7cd70aff1c1e70,

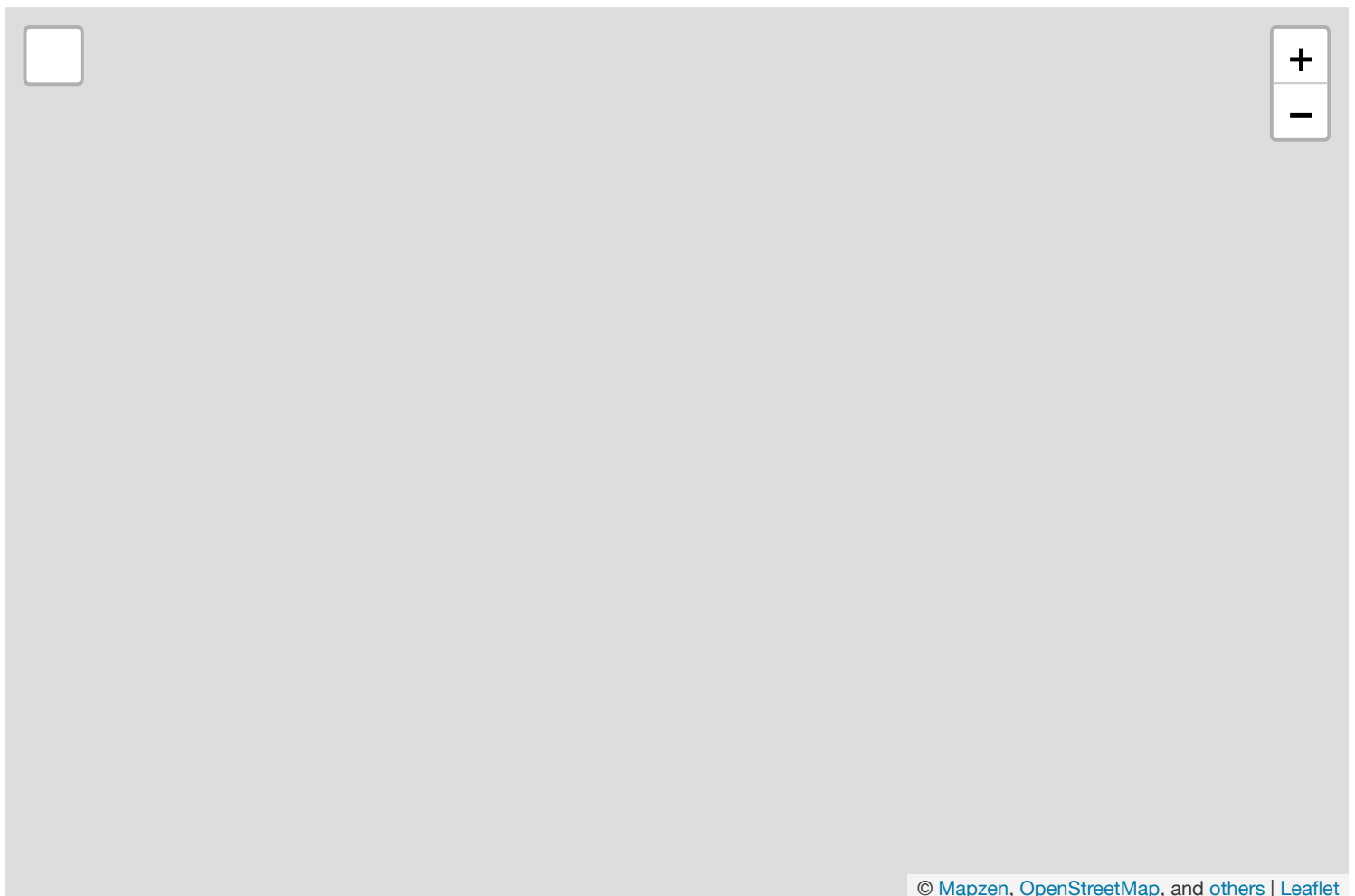
layers:
  _national_park:
    data: { source: _nps_boundary }
    draw:
      lines:
        visible: false
        width: [[8, 0.5px], [18, 5px]]
        color: '#518946'
        order: global.sdk_order_under_roads_1

  _geology:
    data: { source: _nps_geology }
    filter:
      all:
        - { $zoom: { min: 10 } }
        - not: { GLG_SYM: water }
    draw:
      _alpha_polygons:
        order: global.sdk_order_over_everything_but_text_0
        color: |
          function() {
            // Note: this is a block of JavaScript so we can use JS comment s
            var category = feature.GLG_SYM;
            var color = category == 'Qa'      ? '#FFF79A' :
                          category == 'Qb'      ? '#FFF46E' :
                          category == 'Qd'      ? '#fff377' :
                          category == 'Qf'      ? '#dddddd' :
                          category == 'Qp'      ? '#EAC88D' :
                          category == 'Qgdm'     ? '#FCBB62' :
                          category == 'Qgdm(es)' ? '#FEE9BB' :
                          category == 'Qgdm(e)'  ? '#E8A121' :
                          category == 'Qgom(e)'  ? '#EAB564' :

```

```
category == 'Qgom'      ? '#FECE7A' :  
category == 'Qgd'       ? '#FEDDA3' :  
category == 'Qgt'       ? '#FCBB62' :  
category == 'KJmm(c)'   ? '#86C879' :  
category == 'KJm(ll)'   ? '#9FD08A' :  
category == 'JTRmc(o)'  ? '#27BB9D' :  
category == 'TRn'       ? '#ED028C' :  
category == 'TRPMv'     ? '#F172AC' :  
category == 'TRPv'      ? '#F499C2' :  
category == 'PDmt'      ? '#40C7F4' :  
category == 'pPsh'      ? '#9BA5BE' :  
category == 'pDi'       ? '#848FC7' :  
category == 'pDit(t)'   ? '#B28ABF' :  
'#000';  
  
    return color;  
}
```

Reload and marvel:



Pretty!

EXTRA CREDIT

I don't want to take your attention for *granite* (**groan** (<https://www.youtube.com/watch?v=xCc-RWlp7XU>)), but let's make one more change to our map...

Remember our national park boundaries? Let's add those back into our map, so we can see the outlines over our geology polygons.

In your scene file, update the `order` so that it's one above our geology layer and remove the `visible: false` line that we added at the beginning of this post.

```
lines:
  width: [[8, 0.5px], [18, 5px]]
  color: '#518946'
  order: global.sdk_order_over_everything_but_text_1
```

You should see a simple green line outlining the San Juan National Historic Park. Let's make those stand out a little more.

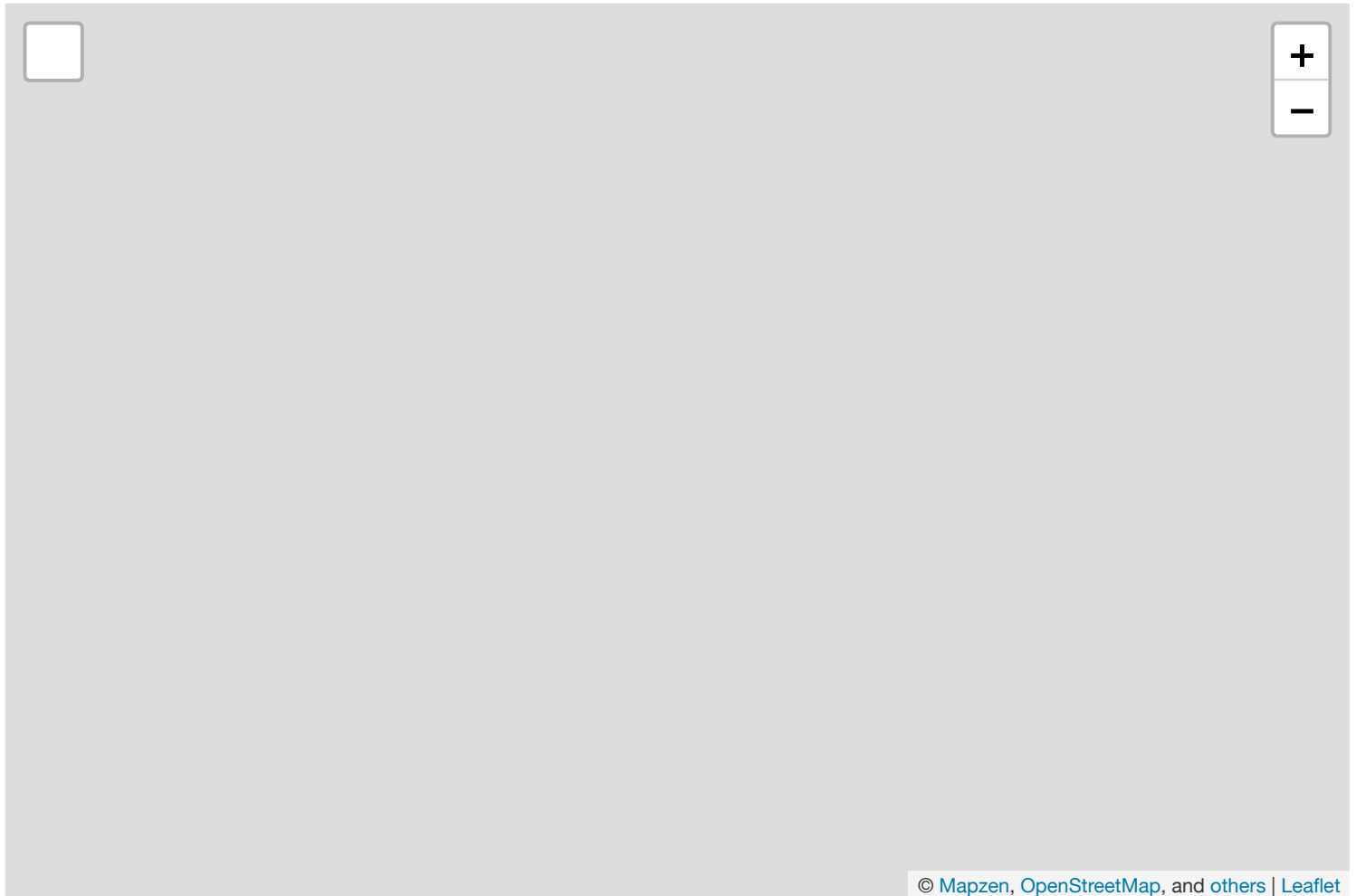
Add another custom style to the top of your scene called `_dashed_lines` :

```
_dashed_lines:
  base: lines
  dash: [3, 1]
  dash_background_color: rgb(149, 188, 141)
```

And update the style name on the national park layer from `lines` to `_dashed_lines` :

```
draw:
  _dashed_lines:
    width: [[8, 0.5px], [18, 5px]]
    color: '#518946'
    order: global.sdk_order_over_everything_but_text_1
```

If you couldn't tell from the code, we changed our plain jane green line into a *dashed* green line. Oooh! The `dash` parameter defines the dash pattern we want to use. Here, our `[3, 1]` creates a dash that is **3** times as long as the line's width, spaced **1** width apart.



The full code for this exercise can be seen on **bl.ocks.org** (<https://bl.ocks.org/rfriberg/bf34f4f0a0505e79c4a8e532587c7c00>). You can also view and modify this scene file in **Tangram Play** (https://mapzen.com/tangram/play/?scene=https%3A%2F%2Fs3.amazonaws.com%2Fmapzen-assets%2Fimages%2Fgeology-map%2Fscene_extra_credit.yaml#12.1816/48.5378/-123.0883).

Thanks for coming along on another whirlwind tour of **mapzen.js** (<https://mapzen.com/documentation/mapzen-js/>) and the **Tangram** (<https://mapzen.com/documentation/tangram/>) scene file. Join us for the next **Make Your Own** (<https://mapzen.com/tag/make-your-own/>) post when we talk about adding our own labels to the map and overriding features in a Mapzen basemap style.

If you have questions or want to show off something you've made with Mapzen, **drop us a line** (<mailto:hello@mapzen.com>). We love to hear from you!

Geologic time scale artwork by Ray Troll, via **trollart.com** (<http://www.trollart.com/fossils1.html>).

~~~

Check out additional tutorials from the **Make Your Own** (<https://mapzen.com/tag/make-your-own/>) series:

- **One Minute Map** (<https://mapzen.com/blog/one-minute-map/>)
- **Map Sandwich** (<https://mapzen.com/blog/map-sandwich/>)
- **Filters & Functions** (<https://mapzen.com/blog/filters-and-functions/>)
- **Put A Label On It** (<https://mapzen.com/blog/tangram-labels/>)
- **Interactive Mapping with Tangram** (<https://mapzen.com/blog/tangram-interactivity/>)
- **Lots of Dots** (<https://mapzen.com/blog/lots-of-dots/>)
- **Customize Your Search** (<https://mapzen.com/blog/customize-your-search/>)

· 14 November 2016 ·



**Rhonda Friberg**

Rhonda is a javascripter who makes maps, trouble, and the occasional pie.

---

© 2017 Mapzen

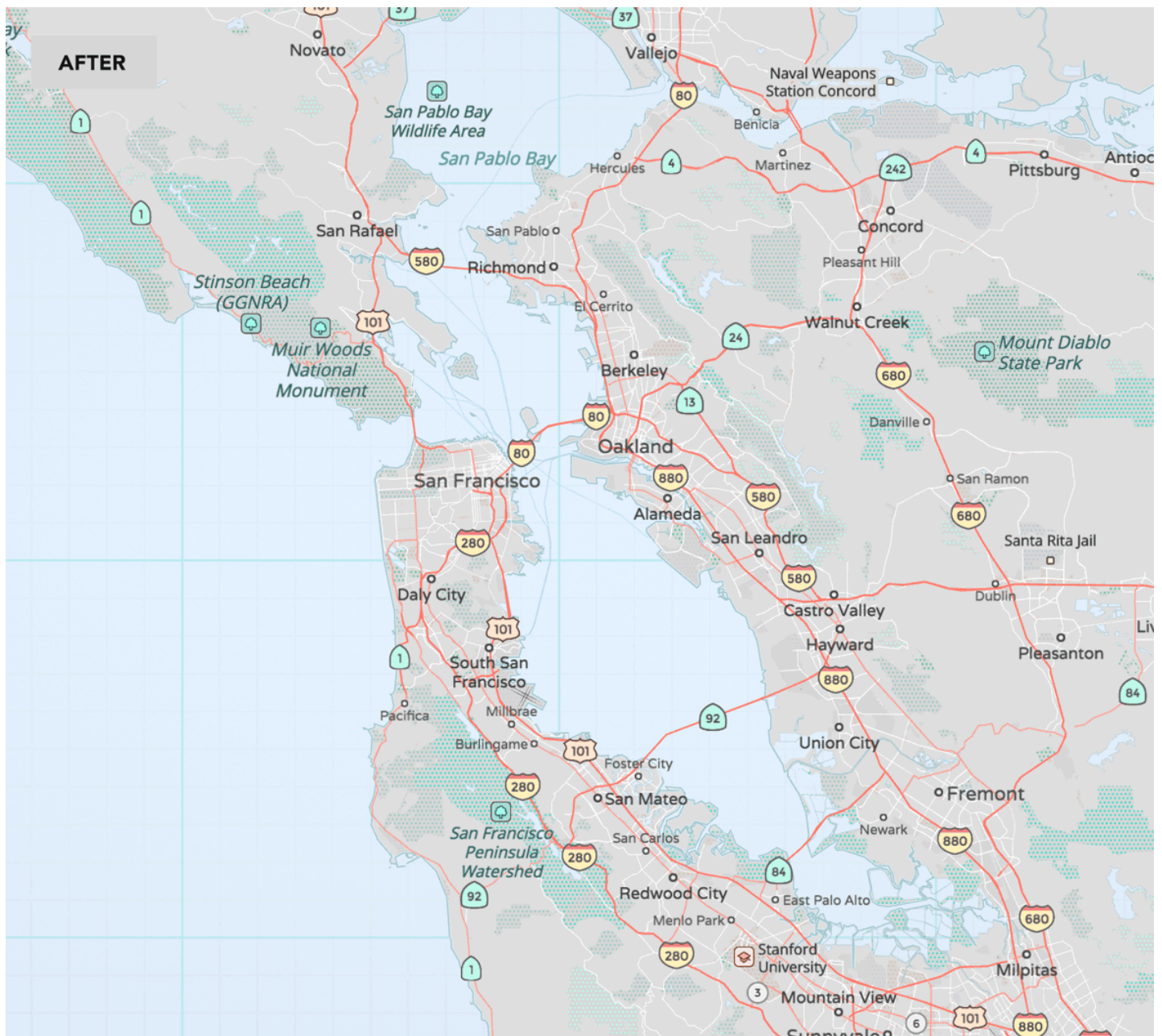
# Road Shields

**cartography** (/tag/cartography) **tangram** (/tag/tangram)

Mapzen house styles have been updated with road shields! This feature greatly enhances map display and is powered by new **Tangram v0.11**

(<https://github.com/tangrams/tangram/releases/tag/v0.11.0>) label placement strategies along with the `network` and `shield_text` properties released in **v1.0 Vector Tiles** (<https://mapzen.com/blog/v1-vector-tile-service/>).

## Bubble Wrap Shields



For this update, we are starting with custom state road shields for California, New York and Pennsylvania (with network values of `US:CA` , `US:NY` , `US:PA` ) along with the US Interstate Highway System ( `US:I` ) and US Federal Routes ( `US:US` ).

Here's how we filter for California shields in Tangram's YAML syntax:



```

US-CA:
  # Match California state highways: `US:CA`
  filter: |
    function() {
      return feature.shield_text &&
        /^US:CA$/ .test(feature.network)
    }
  draw:
    icons:
      sprite: |
        function() {
          return (feature.network + '_' +
            feature.shield_text.length +
            'char');
        }
      text:
        offset: [0px, 1px]
        font:
          fill: [1.0,1.0,1.0]
          size: [[7,7px],[13,9px],[15,10px]]

```

Let's break this down: we select roads with the `US:CA` network tag using a regular expression and do a sprite image lookup in the texture definition based on the number of characters in the `shield_text`. For California Route 1 we'd look for `US:CA_char1`, for California Route 36 we'd look for `US:CA_char2`, and so on. Tangram then composites the `shield_text` onto the sprite artwork dynamically. The same logic holds for US Routes and Interstates.



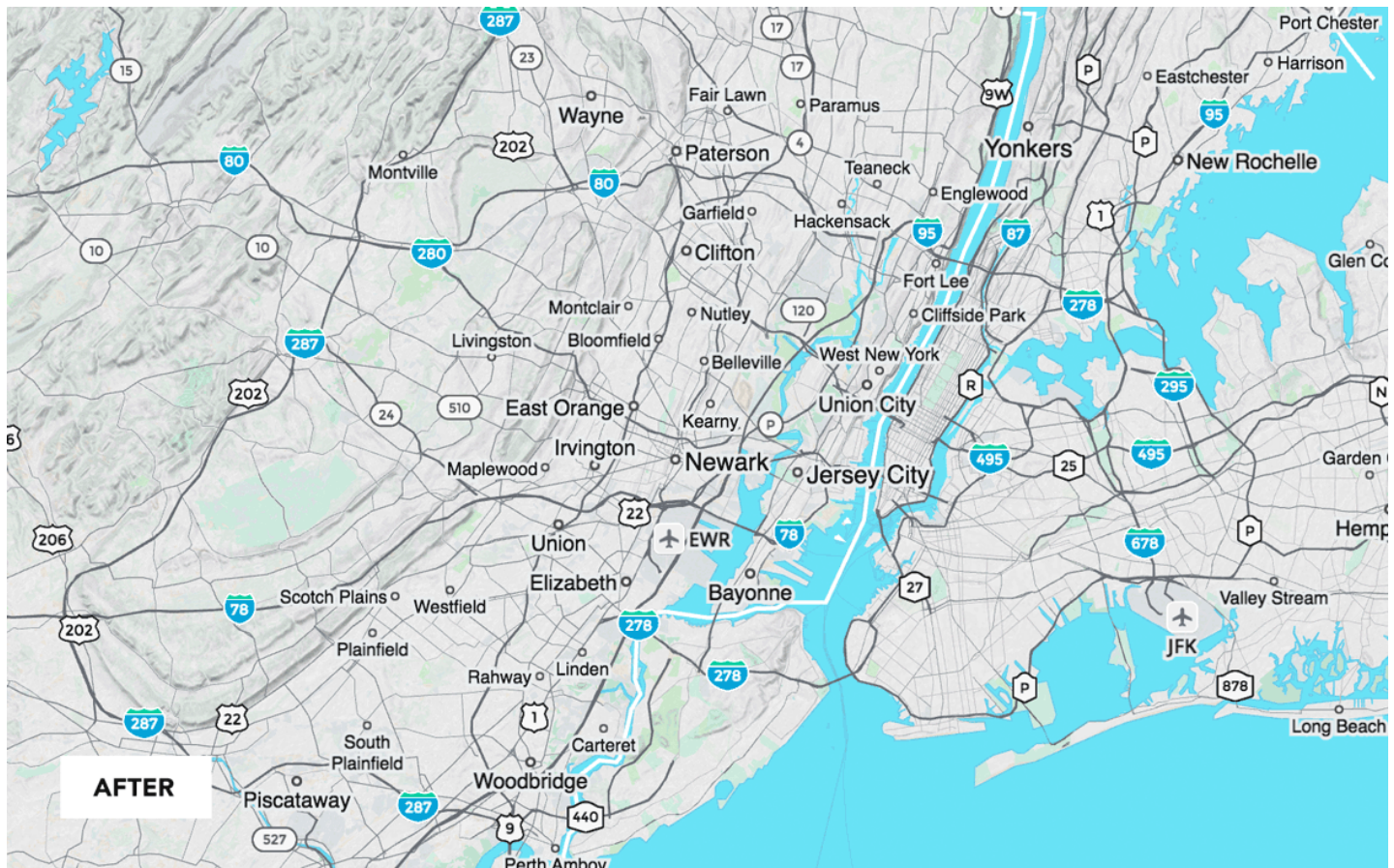
The OpenStreetMap data for road networks in the United States **is very thorough** ([http://wiki.openstreetmap.org/wiki/Interstate\\_Highway\\_relations](http://wiki.openstreetmap.org/wiki/Interstate_Highway_relations)). As more **network** (<http://wiki.openstreetmap.org/wiki/Key:network>) relations are added in Europe and elsewhere it will be easier to support additional custom shields for more countries (and to add logic for when a national road shield should win over an **E-Road** ([https://en.wikipedia.org/wiki/International\\_E-road\\_network](https://en.wikipedia.org/wiki/International_E-road_network)) shield).

Until then, we fall back to generic shield artwork and shield text sourced from the road's basic ref value, and generate generic shields that can handle one to five characters. This helps Europe close the gap, but in the United States if you see a mix of custom shields and generic shields along the same road it's probably because small segments of highway need be added to the overall network relation. But **you can fix this**

([http://wiki.openstreetmap.org/wiki/Interstate\\_Highway\\_relations](http://wiki.openstreetmap.org/wiki/Interstate_Highway_relations))! For reference, you can see how **I-90 through Illinois** (<http://www.openstreetmap.org/relation/104461>) is a relation made up of many **smaller segments of ways** in OSM (<http://www.openstreetmap.org/way/78088815>).)

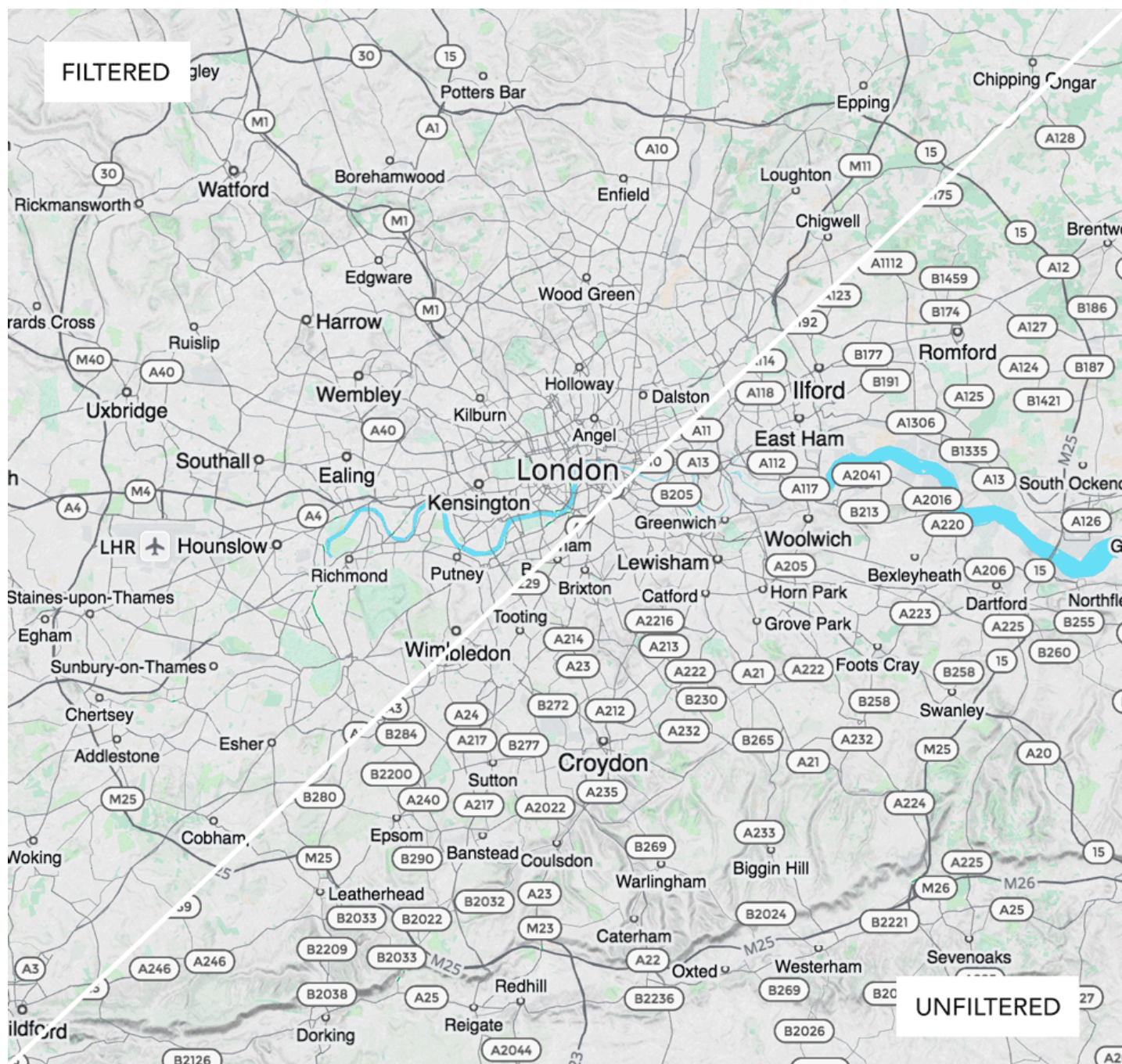
If you have custom shield logic and/or vector artwork to contribute, please open an **issue** ([https://github.com/tangrams/icons/issues/new?title=highway+shield+resources+for+%5Bregion%5D+and/or+%5Bcountry%5D&body=Is+the+highway+network+recorded+in+OSM+as+a+relation?+%60network=XX:YY%60\)+Is+there+public+domain+vector+artwork?](https://github.com/tangrams/icons/issues/new?title=highway+shield+resources+for+%5Bregion%5D+and/or+%5Bcountry%5D&body=Is+the+highway+network+recorded+in+OSM+as+a+relation?+%60network=XX:YY%60)+Is+there+public+domain+vector+artwork?)). (Links to public domain road shield signs are great, but the first step is to ensure that the road network is defined in OpenStreetMap!)

## Walkabout Shields





To improve performance and keep the map balanced, we thin out the total number of shields visible based on the road's importance. Highway shields are visible first, then major roads, and finally minor roads. We also adjusted the relative priority between road shields, road text labels, and other label features.



*Here's London with highway shields filtered on the left, and unfiltered on the right.*

## Tron Shields





Explore all our house styles below:



## Bubble Wrap

( These maps are interactive! Open full screen ↗ (<https://mapzen.com/products/maps/>) )

· 16 November 2016 ·



### Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.



### Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.



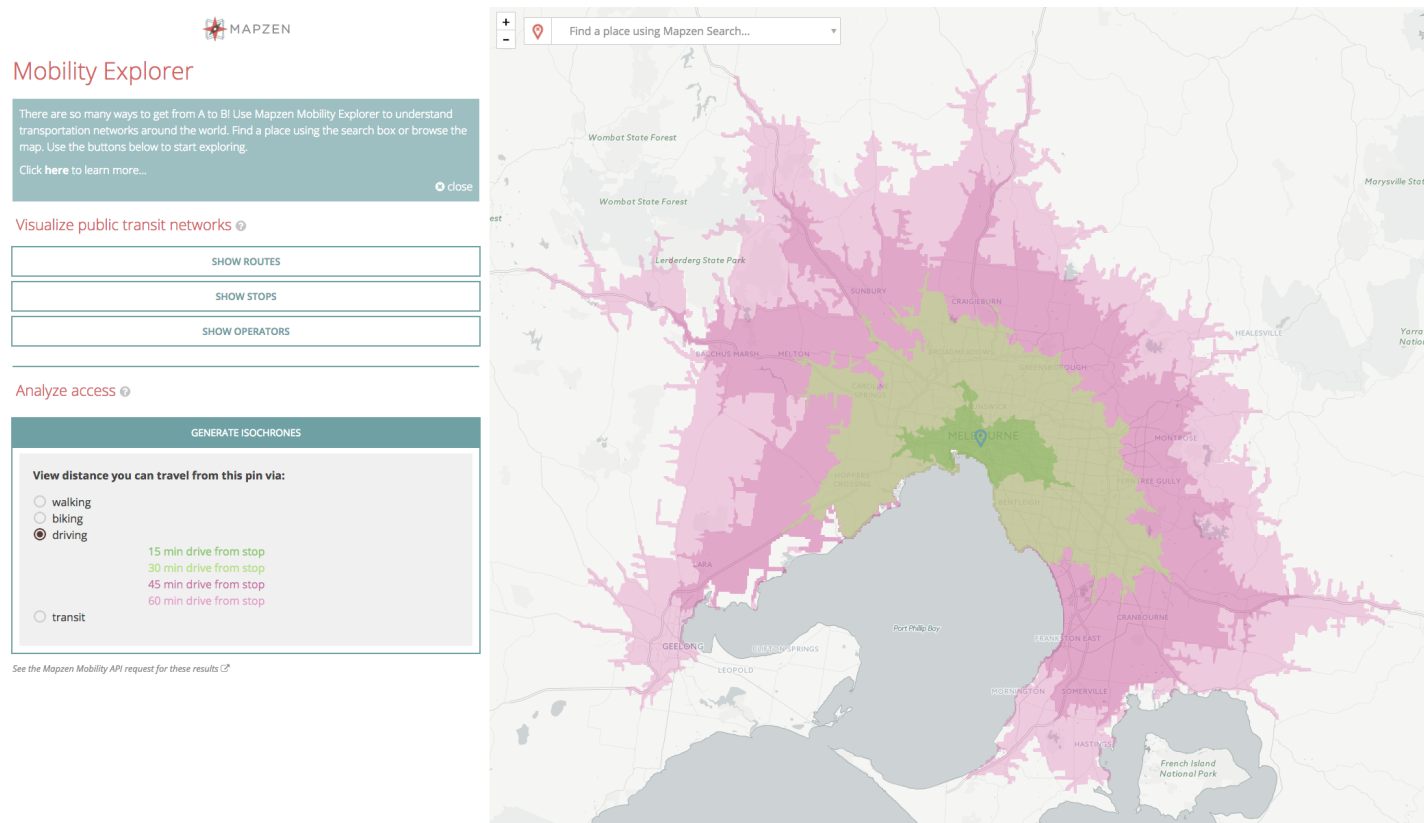


# Range far and wide with Mapzen Isochrone

mobility (/tag/mobility) routing (/tag/routing)

## What are isochrone maps?

Valhalla, Mapzen's routing engine, recently gained the ability to return amazing structures called isochrones. What's an isochrone? The word is a combination of two Greek roots – *iso* means equal and *chrono* means time. So indeed, an isochrone is a structure representing equal time. In our case, it's a line that represents constant travel time about a given location. One can think of isochrone maps as somewhat similar to topographic maps, but instead of lines of constant height, lines of constant travel time are depicted. For this reason other terms common in topography apply, such as contours or isolines.



*Isochrone of Melbourne driving times, via **Mapzen Mobility Explorer***

**(<https://mapzen.com/mobility/explorer/#/isochrones?bbox=144.64393615722656%2C-38.17289287509456%2C145.63201904296875%2C->**



**37.5004656823706&isochrone\_mode=auto&pin=-37.82822612280361%2C144.95728254318237)**

## How are isochrone maps useful?

Isochrone maps can be used to make informed decisions about travel at an individual level and en masse. You can get quantitative answers to questions like:

- What are our lunch options within 5 minutes from here?
- How much of the city lives within walking range of public transit?
- What would adding/removing this road/bus stop/bridge do to travel times?
- Where can I find housing that still has a reasonable commute to the office?

An isochrone map service has a wide variety of use case, from planning departments of DOTs all the way down to consumer applications.

## Technical details

Isochrones are formed in Valhalla by first creating a 2-D grid in `latitude, longitude` about the location. This 2-D grid, or array, is used to define the time or cost it takes to get from the target location to each other grid location. This grid is populated by doing a breadth-first, least-cost first search (basically Dijkstra's) from the origin location. At each iteration, the grid cells that are touched by a road segment or graph edge are marked with the time and cost from the origin, if less than the currently marked time. Once the expansion of the graph exceeds the maximum isochrone contour time, the grid-marking process terminates. This leaves a 2-D grid that has the time or cost to reach each grid location. It can be used to provide a very fast (but approximate) way to query the time/distance for many locations about an origin. Ultimately this could be a way to do very large one-to-many matrices. At this time we do not return the 2-D array of times, but this is a possibility in the future.

Instead, the 2-D grid is used to find the isochrone contours by using a well-known **contouring** (<http://paulbourke.net/papers/conrec/>) method developed by Paul Bourke in the 1980s. This method finds grid cells that have neighbors with values that lie on opposing sides of the contour value. (Another way of looking at it: the current cell has a time value above the contour value and a neighbor has a time value below the contour value.) The contouring algorithm generates line segments between adjacent grid cells based on several possible cases. The result is a set of 2 point line segments for each.

We are then left with a minimization problem where we want to chain together as many adjacent 2 point pieces as possible. This gets a bit tricky because the segments follow no specific order or direction, meaning you may have to search the whole list to find a pair of adjacent segments, and once you do, you may have to switch the order of the points to append or prepend it to a chain. We end up hashing the endpoints of each segment so we can easily find which ones are adjacent. In the end we are left with the minimum number of lines representing each contour value.

## Rendering

Raster-based rendering of isochrones is by far the most common method of display. This presents a bit of a problem in that you don't have much control over what appears in the raster (each pixel), so you've got to convert the raster to something you can display in your preferred style. Because of this, while internally compute isochrones using a raster datatype, we currently return vectorized data.

We currently support `linestring` and `polygon` primitive types **in the API** (<https://mapzen.com/documentation/mobility/isochrone/api-reference/#inputs-of-the-isochrone-service>). Rendering `linestring` data is **quite straight-forward** (<https://mapzen.com/tangram/play/?scene=https%3A%2F%2Fmapzen.com%2Fapi%2Fscenes%2F22%2F367%2Fresources%2Fdefault.yaml#12.9543/19.4147/-99.1362>).



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

[Learn more.](#)

*(Mexico City driving isochrones  
20 min, 15 min, 10 min, 5 min)*

Rendering polygons is tricky business. When working with isochrones its generally useful to see them in the presence of a map. So to be able to see the map one must render the polygons as semi-transparent (unless you've got a **vector based map like Tangram where you could add the isochrone polygons into a map sandwich** (<https://mapzen.com/tangram/play/?scene=https://mapzen-assets.s3.amazonaws.com/resources/isochrones-mexico-city-polygons.yaml#13.2337/19.4313/-99.1421>)). Because isochrones are actually contours, the polygons returned will be necessarily nested. This nesting causes overlapping transparent polygons, which leaves you with portions of the map that have blended colors from two different isochrones and even worse, varying amounts of transparency. To solve this we need a

couple of things – the geometry of the polygons has to be free of degeneracies (say self-intersections), and holes in polygons have to be properly represented. These are not currently implemented, but we're working on it.

Consider your use-case, read the docs linked below, and use your best judgement to determine if rendering isochrones as linestrings or polygons is right for you!

## Where is it?

You can find the **API documentation here**

(<https://mapzen.com/documentation/mobility/isochrone/api-reference/>) and you see a live demo of it right **here** (<https://mapzen.com/products/isochrone/>). Isochrones are also a key part of **Mapzen Mobility Explorer** (<https://mapzen.com/blog/introducing-mobility-explorer>). Let us know what you think!

· 17 November 2016 ·



### Kevin Kreiser

Kevin works on routing at Mapzen but secretly tries to work in all mapping disciplines. Er iss aa Pennsilfaanisch Deitscher.



### David Nesbitt

Dave leads Mapzen Mobility engineering. Rides a variety of 2 wheel vehicles.

---

© 2017 Mapzen

# Introducing Mapzen Mobility Explorer

**mobility** (/tag/mobility) **transitland** (/tag/transitland) **routing** (/tag/routing)

Mapzen is excited to introduce **Mobility Explorer** (<https://mapzen.com/mobility/explorer>): a new tool for querying and visualizing **Transitland** (<https://transit.land/>) data and Mapzen Mobility services. Mobility Explorer makes it easy to see relationships between transit datasets, analyze travel options, and share your discoveries.

Mobility Explorer is built for developers of maps and apps, public policy advocates and planning experts, transit nerds, and anyone who is curious about exploring transit data from the top down, bottom up, or starting anywhere in between. Our goal is to make existing transit data more accessible by presenting it visually, and sharing it more readily.

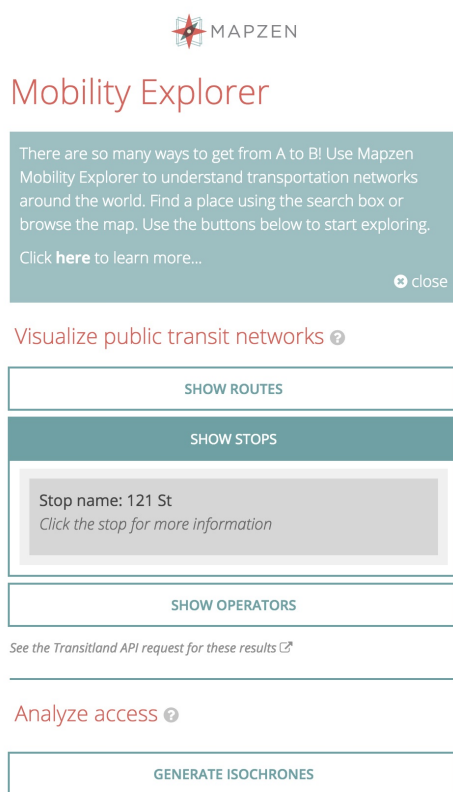
There are so many things you can do with Mobility Explorer. Even if you aren't setting out to create a map or app, you might want to understand the connections between transit systems in your own neighborhood. Perhaps you work for a transit operator, and want to visually verify the data Transitland has for your agency. You might be visiting a neighborhood in a new city and want to see all of the routes that pass through it, and where you can go from your new 'hood. Or, maybe you *are* looking to get creative and develop a new web app, but want to see what data exists in the Transitland Datastore before you begin.

Mobility Explorer provides an approachable way to query for and immediately visualize the data. Search for points of interest in the search box, or drop a pin on the map to set your bounding box, then use the Transitland Data buttons to view the stops, routes, and operators serving the area. Each option you select is saved as part of the URL, creating a shareable link that leads to the view of the data that you've created.

## Connections between datasets

One goal for the Explorer is to highlight the connections that exist between transit datasets. Let's look at an example; say you are in an Empire State of Mind, and feel like learning more about transit in New York City. You can **use the Explorer to zoom in on the Richmond Hill neighborhood** (<https://mapzen.com/mobility/explorer/#/stops?>

**bbox=-73.84803771972656%2C40.68861108584693%2C-73.79310607910156%2C40.71145106322093)** and click the **SHOW STOPS** button to see what's there.



**MAPZEN**

## Mobility Explorer

There are so many ways to get from A to B! Use Mapzen Mobility Explorer to understand transportation networks around the world. Find a place using the search box or browse the map. Use the buttons below to start exploring. Click [here](#) to learn more...

Visualize public transit networks ?

**SHOW ROUTES**

**SHOW STOPS**

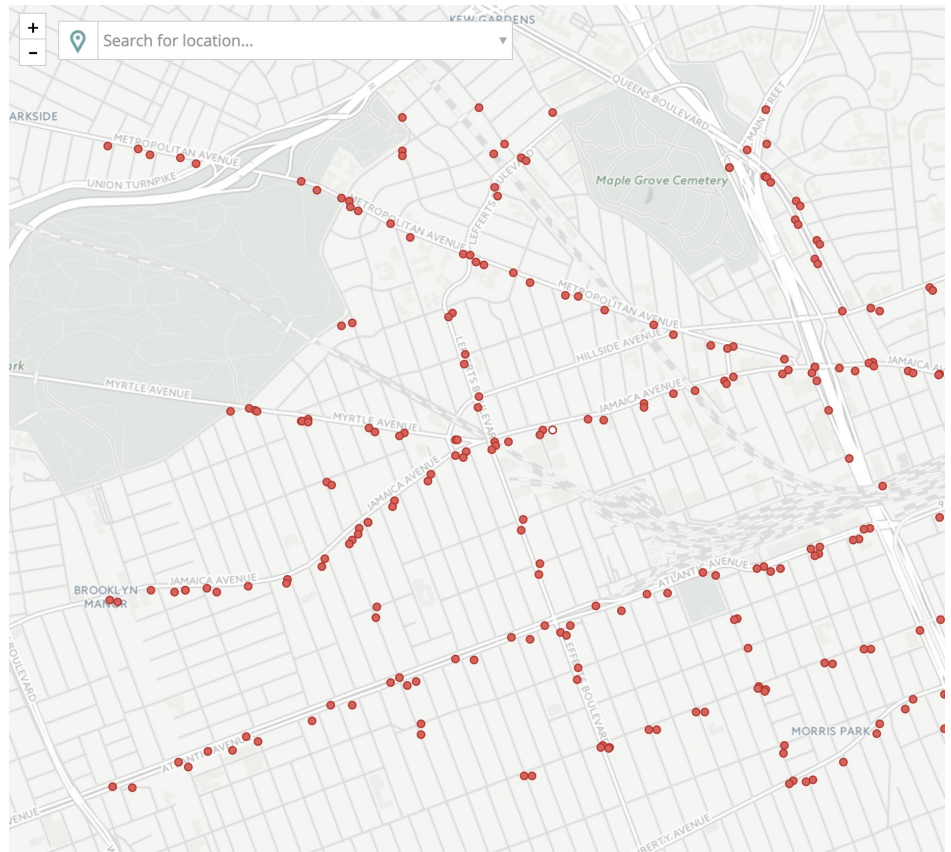
Stop name: 121 St  
Click the stop for more information

**SHOW OPERATORS**

See the Transitland API request for these results ?

Analyze access ?

**GENERATE ISOCHRONES**



(<https://mapzen.com/mobility/explorer/#/stops?bbox=-73.84803771972656%2C40.68861108584693%2C-73.79310607910156%2C40.71145106322093>)

You can get information on a **single stop** ([https://mapzen.com/mobility/explorer/#/stops?bbox=-73.84803771972656%2C40.68861108584693%2C-73.79310607910156%2C40.71145106322093&onestop\\_id=s-dr5rxcrs1-121st](https://mapzen.com/mobility/explorer/#/stops?bbox=-73.84803771972656%2C40.68861108584693%2C-73.79310607910156%2C40.71145106322093&onestop_id=s-dr5rxcrs1-121st)) by hovering over the stops to view their name, and click them to get detailed information.

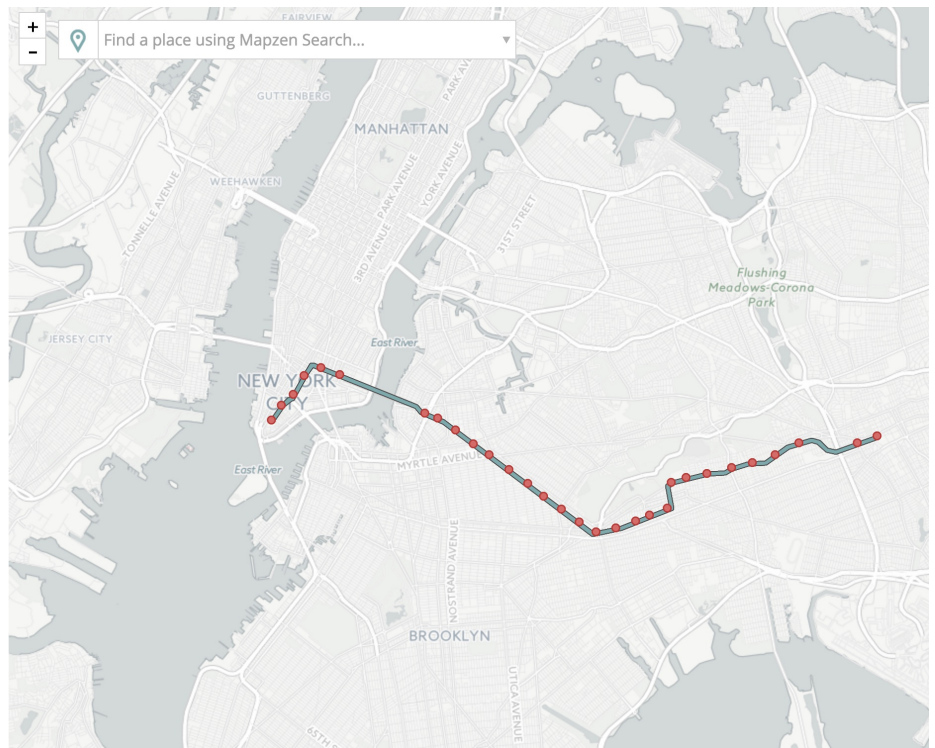
After clicking the 121st Street stop, you see that it is served by the **JZ** (<https://vimeo.com/14219280>) line. Once you select the route you can opt to view all of the stops—it's not 99 stops, but it visits quite a few.

Visualize public transit networks ?

SHOW ROUTES

**Route name:** J  
**Onestop ID:** r-dr5r-j ?  
**Vehicle type:** metro  
**URL:** <http://web.mta.info/nyct/service/pdf/tjcur.pdf>  
**Route description:** Trains operate weekdays between Jamaica Center (Parsons/Archer), Queens, and Broad St, Manhattan. During weekdays, Trains going to Manhattan run express in Brooklyn between Myrtle Av and Marcy Av from about 7 AM to 1 PM and from Manhattan from 1:30 PM and 8 PM. During weekday rush hours, trains provide skip-stop service. Skip-stop service means that some stations are served by J trains, some stations are served by the Z trains, and some stations are served by both J and Z trains. J/Z skip-stop service runs towards Manhattan from about 7 AM to 8:15 AM and from Manhattan from about 4:30 PM to 5:45 PM.  
**Long name:** Nassau St Local  
**Line color:** 996633  
**Operated by:** MTA New York City Transit  
**Operator Onestop ID:** o-dr5r-nyct

☒ Show stops served by route  
[Show route stop patterns ?](#)



([https://mapzen.com/mobility/explorer/#/routes?bbox=-74.03205871582031%2C40.614994915836924%2C-73.59260559082031%2C40.79769722250925&onestop\\_id=r-dr5r-j](https://mapzen.com/mobility/explorer/#/routes?bbox=-74.03205871582031%2C40.614994915836924%2C-73.59260559082031%2C40.79769722250925&onestop_id=r-dr5r-j))

You can also see that this route is operated by MTA. By clicking on the operator name, you are taken to a view of the operator's entire service area. From there, you can choose to view **all stops and routes served by the operator**

([https://mapzen.com/mobility/explorer/#/routes?bbox=-74.46395874023438%2C40.317231732315236%2C-73.0535888671875%2C41.04828819952275&operated\\_by=o-dr5r-nyct](https://mapzen.com/mobility/explorer/#/routes?bbox=-74.46395874023438%2C40.317231732315236%2C-73.0535888671875%2C41.04828819952275&operated_by=o-dr5r-nyct)).



## Mobility Explorer

There are so many ways to get from A to B! Use Mapzen Mobility Explorer to understand transportation networks around the world. Find a place using the search box or browse the map. Use the buttons below to start exploring.

Click [here](#) to learn more...

close

Visualize public transit networks ?

SHOW ROUTES

353 routes

Hover over a route line for more information

Operated by: o-dr5r-nyct

Style routes by: ?

☐ Mode

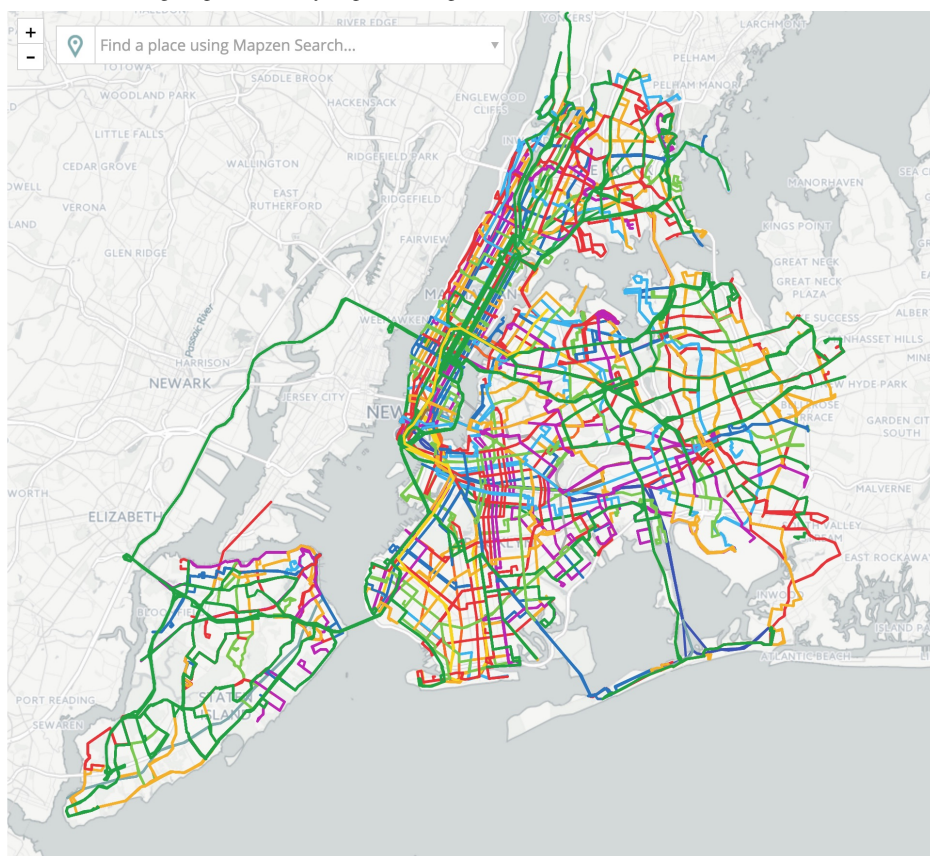
SHOW STOPS

SHOW OPERATORS

See the Transitland API request for these results ↗

Analyze access ?

GENERATE ISOCHRONES



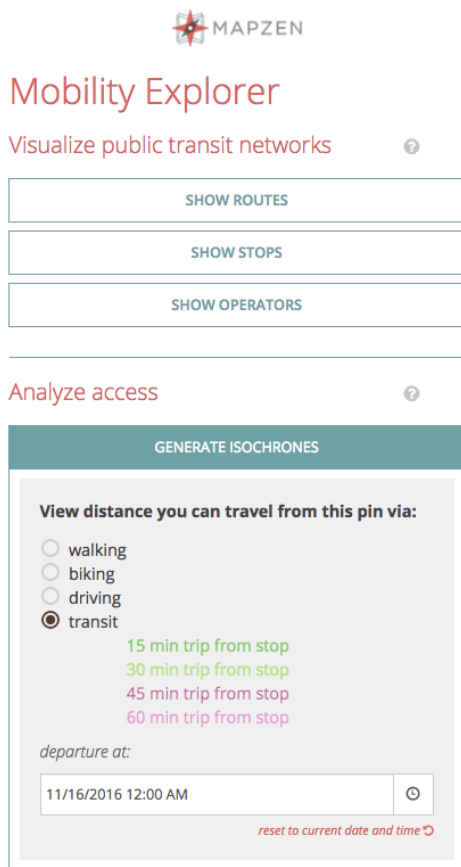
([https://mapzen.com/mobility/explorer/#/routes?bbox=-74.46395874023438%2C40.317231732315236%2C-73.0535888671875%2C41.04828819952275&operated\\_by=o-dr5r-nyct](https://mapzen.com/mobility/explorer/#/routes?bbox=-74.46395874023438%2C40.317231732315236%2C-73.0535888671875%2C41.04828819952275&operated_by=o-dr5r-nyct))

## Mobility Explorer with isochrones

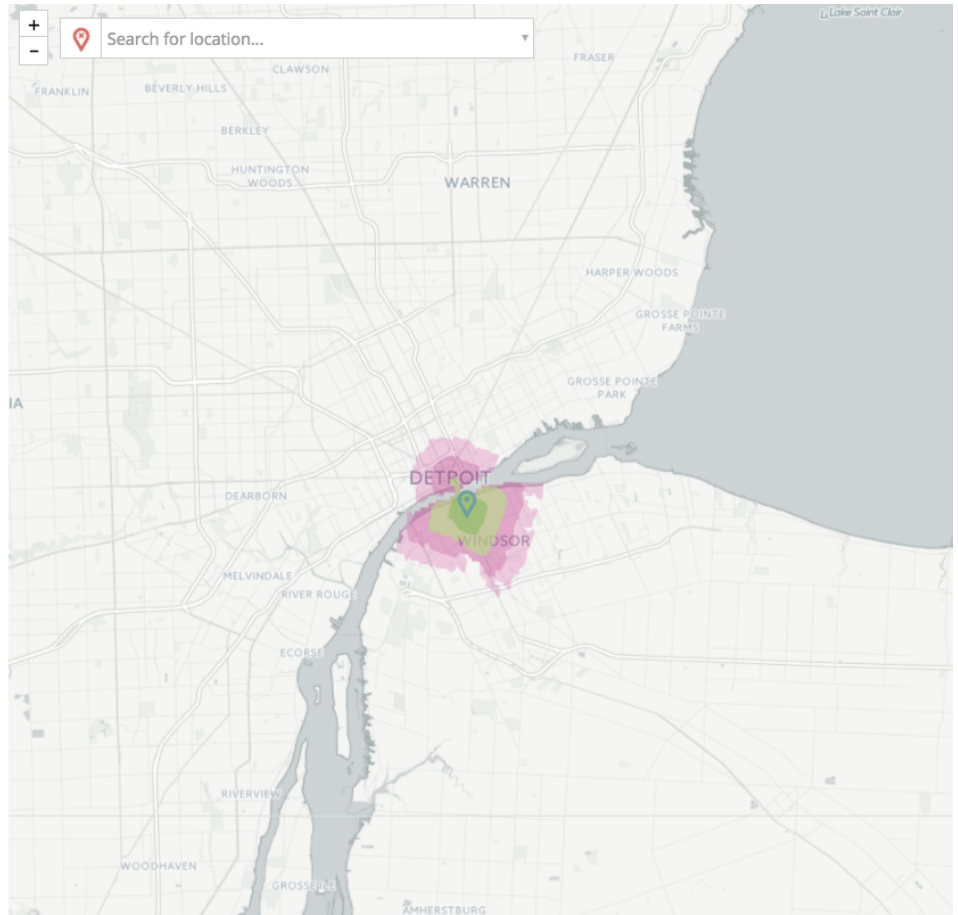
Another goal for the Explorer is to use Mapzen Mobility services to analyze and understand travel options beyond fixed-route transit. And so, Explorer also presents a brand-new service: the **Mapzen Isochrone** (<https://mapzen.com/blog/introducing-isochrone-service>) service. Isochrones, often referred to as “sheds”, tell you where you can travel from a particular point at various time intervals.

Let’s say you are a city boy, born and raised in South Detroit. You’re interested in taking a midnight train going anywhere, but you’ve only got an hour. Just how far could you get? Head over to the Explorer and drop a pin (or enter your search terms) to find out! You can then choose to view isochrones for walking, biking, driving, or transit. The transit option also allows you to choose the time of departure, since the distance you can travel will vary with changes in service throughout the day. So, here’s the **transit isochrone for this point in South Detroit** ([https://mapzen.com/mobility/explorer/#/isochrones?bbox=-83.22246551513672%2C42.18299552873959%2C-82.6113510131836%2C42.43333965406586&departure\\_time=2016-11-](https://mapzen.com/mobility/explorer/#/isochrones?bbox=-83.22246551513672%2C42.18299552873959%2C-82.6113510131836%2C42.43333965406586&departure_time=2016-11-)

**17T00%3A00&isochrone\_mode=multimodal&pin=42.31260817230085%2C-83.03466796874999)** (er, Windsor, Ontario  
**(http://www.mlive.com/entertainment/detroit/index.ssf/2012/01/steve\_perry\_finally\_answers\_th.html))** with the departure time set for midnight:



See the Mapzen Mobility API request for these results [↗](#)



**(https://mapzen.com/mobility/explorer/#/isochrones?bbox=-83.22246551513672%2C42.18299552873959%2C-82.6113510131836%2C42.43333965406586&departure\_time=2016-11-17T00%3A00&isochrone\_mode=multimodal&pin=42.31260817230085%2C-83.03466796874999)**

Looks like you can't get too far in an hour at this time of night, but now that you can visualize this on a map and share your map view using its unique URL, you could start to lobby for improved transit options—so don't stop believin'!

## Try out Mobility Explorer

We hope Mobility Explorer helps you discern the richness of data in Transitland and highlights the connections between routes and stops of different operators.

As we expand our range of Mapzen Mobility services, you'll be able to use them through both an API and through Mobility Explorer. We can't wait to see what we learn together!

Try Mobility Explorer at **<https://mapzen.com/mobility/explorer>**  
(**<https://mapzen.com/mobility/explorer>**).

Happy Exploring!

*Preview image: "Textile, Traffic, 1983" by Jurgen Lehl, **via Cooper Hewitt***  
(**<https://collection.cooperhewitt.org/objects/18801037/images/>**)

· 17 November 2016 ·



### **Meghan Hade**

Meghan is a software engineer with a background in urban planning on Mapzen's Mobility team. She works in javascript and plays in calligraphy, block printing, and finding ways to stash n+1 bikes in a San Francisco apartment.

---

© 2017 Mapzen

# It's our 2 year anniversary – Announcing Valhalla 2.0!

**routing** (/tag/routing) **mobility** (/tag/mobility)

It is hard to believe that we have been developing the open-source routing and navigation software Valhalla for 2 years now! It is also extremely gratifying to see that after 2 years, the primary design tenets behind Valhalla still hold true. Tile-based data, use of dynamic, run-time costing, emphasis on guidance and mobile use, and so much more...

## Highlights of our past work

**Valhalla Open Source Routing** (<https://mapzen.com/blog/valhalla-intro/>)

**Why Tiles?** ([https://mapzen.com/blog/valhalla-why\\_tiles/](https://mapzen.com/blog/valhalla-why_tiles/))

**Introducing Valhalla** (<https://mapzen.com/blog/introducing-valhalla/>)

**Dynamic Costing** (<https://mapzen.com/blog/dynamic-costing-via-sif/>)

**Elevation** (<https://mapzen.com/blog/moving-on-up/>)

**Valhalla Bicycle Route Options** (<https://mapzen.com/blog/valhalla-bicycle-routing-options/>)

**Elevation-Influenced Bicycle Routing** (<https://mapzen.com/blog/making-the-grade-elevation-influenced-bicycle-routing/>)

**Shaping Up Your Route Guidance** (<https://mapzen.com/blog/narrative-shape-up/>)

**Transit Routing** (<https://mapzen.com/blog/transit-routing/>)

**Voice Guidance** (<https://mapzen.com/blog/voice-guidance/>)

**Speed Tiles and Traffic-Influenced Routing** (<https://mapzen.com/blog/speed-tiles/>)

**Country Specific Access Restrictions** (<https://mapzen.com/blog/access-denied/>)

**Time-Distance Matrix** (<https://mapzen.com/blog/matrix/>)

**Optimized Routing** (<https://mapzen.com/blog/optimized-route/>)

**Customize Mapzen Turn-by-Turn Narrative Language (<https://mapzen.com/blog/narrative-language/>)**

**DIY Valhalla (<https://mapzen.com/blog/a-routing-engine-of-ones-own/>)**

**Mapzen Isochrone (<https://mapzen.com/blog/introducing-isochrone-service/>)**

## **Valhalla 2.0**

This week we have designated the latest packaged release as Valhalla 2.0. (This is transparent to users of Mapzen Turn-By-Turn service and all the Mobility products – users do not need to make any changes other than to enjoy faster responses and more features!)

We made several key enhancements to Valhalla, in particular to its tiled data. These changes include:

- Update the highway hierarchy such that edges are only stored on the hierarchy level they are assigned to. Motorways, for instance, are now represented only on the highway hierarchy. In prior versions of Valhalla, motorways existed on each hierarchy: highway, arterial, and local. This change decreases data size and facilitates several new features: complex, via-way restrictions and traffic integration.
- Support memory mapping of tar'd Valhalla graph tile extracts. This improves caching and concurrent performance. The option of using individual graph tiles as files that are read into cache still exists for deployments where tighter control of memory use is required.
- Redesigned several data structures within the graph tiles to allow for future growth without breaking backward-compatibility. Slots for additional attribution and data structures have been added to the tile definition and spare fields within existing structures will allow incremental additions.
- Development of new features continues. Isochrones are officially supported with this release. Map-matching features are in development and should be ready soon.

## **The Valhalla Team**





So what does the development team have to say about this milestone?

**Dave Nesbitt**

*"It has been fun to see the software develop through time and grow-up. I think we now have a nearly full set of features to support routing, navigation, and analysis applications. I am particularly excited to see where we can take this in the mobile and embedded navigation space."*

**Duane Gearhart**

*"I am proud of the open services that we have created and the partnerships we have formed over the past two years. I am looking forward to extending mobile navigation and improving our services including user accessibility."*

**Greg Knisely**

*"I am very excited to see the use of complex (via-way) restrictions within the path finding algorithms. This will greatly enhance the user experience by properly using turn restrictions that were previously not included in the path finding. Valhalla 2.0 will enable us to continue to enhance our products and services and in turn provide more accurate results to our customers."*

**Kevin Kreiser**

*"Man muss nicht müssen... wir aber doch!"*

## Kristen DiLuca

*"Amazing...so many accomplishments in such a short period of time! I'm so proud of the work that has gone into making our products and services. Being part of the open community really allows us to build awesome routing and navigation software, while listening and responding effectively to the needs of our users."*

Check out the **Mapzen Turn-by-Turn documentation**

(<https://mapzen.com/documentation/turn-by-turn/>) and **let us know if you have any questions** (<https://twitter.com/intent/tweet?text=@Mapzen%20@ValhallaRouting%20Hi!>).

· 17 November 2016 ·



### David Nesbitt

Dave leads Mapzen Mobility engineering. Rides a variety of 2 wheel vehicles.



### Duane Gearhart

Duane is a software engineer @mapzen specializing in quality route guidance and real-world route analysis.



### Kevin Kreiser

Kevin works on routing at Mapzen but secretly tries to work in all mapping disciplines. Er iss aa Pennsilfaanisch Deitscher.



### Kristen DiLuca

Kristen is a software engineer specializing in our routing API services. She also enjoys dabbling in javascript for our open source routing test tools and always welcomes new challenges.



### Greg Knisely

Greg is a data mungers and software engineer for Mapzen's routing software.



© 2017 Mapzen

# वैलहलल नलरुडशन अल हलंदी डें उडललडुड!

routing (/tag/routing) data (/tag/data) osm (/tag/osm)

## नडुडुडर • Namaskār

At Mapzen we are lucky to have people who speak a wide variety of languages on the team and the support of a vibrant open source community. Before today, Valhalla, our open source routing engine had six **languages** (<https://github.com/valhalla/odin/tree/master/locales>) - Czech, English, French, German, Italian and Spanish. Today we have added हलंदी (Hindi) 🎉 making it seven and another step closer to a more universal routing engine! This means you can now build applications with written and spoken routing instructions in 7 different languages! (And shiver me timbers! We also have special support **for pirates** (<https://mapzen.com/blog/narrative-language/>).)

Here are Hindi navigation instructions in action on our Android app, EraserMap, **along Bistupur Road in Jamshedpur, Jharkhand** (<https://tangrams.github.io/bubble-wrap/#16.50849/22.79798/86.18448>):

*EraserMap app on Android with instructions in Hindi.*

**Download it for free on the Play Store! (<https://play.google.com/store/apps/details?id=com.mapzen.erasermap>)**

### **What's so special about Hindi you ask?**

Hindi is spoken as a first language in northern and central India by more than 258 million people and by another 9 million in Nepal. The Hindi diaspora numbers in the millions: Mauritius, South Africa, Yemen, Uganda, the United States and Canada all have more than 100,000 speakers each. Such a wide distribution makes Hindi one of the most spoken languages of the world, and personally important to Varun and me.

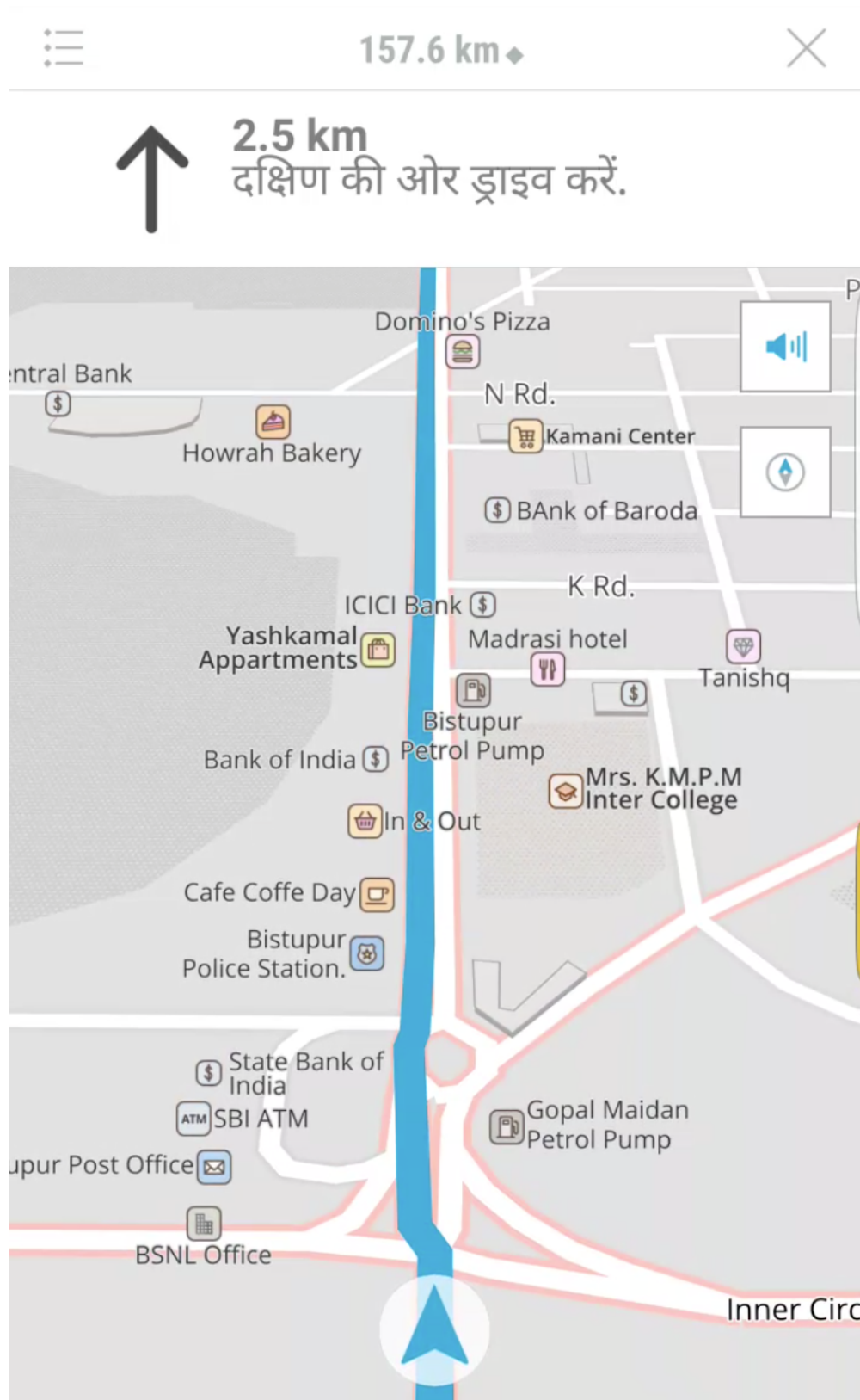
**Thejesh GN** (<https://twitter.com/thej>) wrote a great post (<http://thejeshgn.com/2016/09/30/where-are-the-indian-language-maps/>) about the localized and regional languages in India and their coverage and representation in OpenStreetMap. No mapping provider currently offers routing guidelines for all Indian regional languages. Google currently offers navigation in Hindi and **MapMyIndia** (<http://www.mapmyindia.com/>) offers a handful of regional languages for subscribers. We'd love to see more languages available as open data, and we'll show you how you can do it!

## Picking the right words

**Voice matters** (<https://mapzen.com/blog/voice-guidance/>). How to choose the words and the tone of the navigation instructions? First, a little bit about Hindi. It is considered to be a Category II language in terms of difficulty for speakers of English<sup>1</sup>. The word order is typically subject-object-verb as compared to English which has subject-verb-object. It draws much of its vocabulary from Sanskrit and is a highly inflected language which utilizes prefixes and suffixes to form words and to express grammatical relations. To give an example, second-person personal pronouns are marked for three levels of politeness:

- singular form 'tu (तू)': informal, extremely intimate, or disrespectful
- singular form 'tum (तुम)': informal and showing intimacy
- plural form 'aap (आप)': formal and respectful

For our navigation instructions, we decided to use the formal and respectful tone: 'आप अपने गंतव्य पर पहुंच चुके हैं' instead of an informal one: 'तुम अपने गंतव्य पर पहुंच गए हैं'.



A lot our discussions ended up weighing a more accurate Hindi translation against what works for a navigation system and scalability. For example, when looking at terms for lengths we started with translating every measure like '1/10th of a mile' to 'एक मील का दसवां' and 'less than 10 feet' to 'कम से कम 10 फुट'. These translations, though accurate, were not easy to scale to every possibility of a fraction, and 'less than 10 feet' in Hindi is very different from 'in a little over 10

feet'. The interesting thing we realized was that when speaking in Hindi, especially in India, people will often mix in English words and numbers. A lot of the words in Hindi have come from English. So to make these sentence structures adaptable and easy to scale to different combinations of fractions and measures, we decided to go with 'मील का 1/10 हिस्सा' for '1/10th of a mile'.

Hindi, like French and Spanish, uses grammatical gender. This got really tricky when constructing phrases that might contain street names that could be either masculine or feminine. (India has most of its street and road named after politicians, freedom fighters and Bollywood celebrities). Also, in most cases the words 'road', 'street' or 'marg' (which means tertiary road) are included within the street name and are hard to separate out.

For example, we debated the usage of the linking verb 'becomes' as feminine (जाती) or masculine (जाता):

"Vine Street becomes Middletown Road." > "Vine Street Middletown Road बन जाती है."

OR

"Mahatma Gandhi Marg becomes Link Road" > "Mahatma Gandhi Marg Link Road बन जाता है."

Our efforts are not limited to routing. We recently wrote about **supporting Indian languages in our map rendering engine, Tangram** (<https://mapzen.com/blog/languages-of-india/>).

Valhalla is a 100% open source routing engine and we want to invite YOU, our language experts around the world, to add language(s) for routing instructions, or even make changes and help us improve existing ones. We really want to make a better and more intuitive routing experience for everyone, no matter what language they speak.

## How to contribute your own language

To get started, go to **Valhalla/ODIN repository** (<https://github.com/valhalla/odin/tree/master/locales>) and follow the instructions in the README. A few guidelines:

- Do not translate the JSON keys or phrase tags. An example using the ramp instruction:

The **JSON keys** and the **phrase tags** are highlighted in **red** below  
Please **do not** translate these items

```
"ramp": {
  "phrases": {
    "0": "Take the ramp on the <RELATIVE_DIRECTION>.",
    "1": "Take the <BRANCH_SIGN> ramp on the <RELATIVE_DIRECTION>.",
    "2": "Take the ramp on the <RELATIVE_DIRECTION> toward <TOWARD_SIGN>.",
    "3": "Take the <BRANCH_SIGN> ramp on the <RELATIVE_DIRECTION> toward <TOWARD_SIGN>.",
    "4": "Take the <NAME_SIGN> ramp on the <RELATIVE_DIRECTION>.",
    "5": "Turn <RELATIVE_DIRECTION> to take the ramp.",
    "6": "Turn <RELATIVE_DIRECTION> to take the <BRANCH_SIGN> ramp.",
    "7": "Turn <RELATIVE_DIRECTION> to take the ramp toward <TOWARD_SIGN>.",
    "8": "Turn <RELATIVE_DIRECTION> to take the <BRANCH_SIGN> ramp toward <TOWARD_SIGN>.",
    "9": "Turn <RELATIVE_DIRECTION> to take the <NAME_SIGN> ramp."
  },
  "relative_directions": ["left", "right"],
  "example_phrases": {
    "0": ["Take the ramp on the left.", "Take the ramp on the right."],
    "1": ["Take the I 95 ramp on the right."],
    "2": ["Take the ramp on the left toward JFK."],
    "3": ["Take the South Conduit Avenue ramp on the left toward JFK."],
    "4": ["Take the Gettysburg Pike ramp on the right."],
    "5": ["Turn left to take the ramp.", "Turn right to take the ramp."],
    "6": ["Turn left to take the PA 283 West ramp."],
    "7": ["Turn left to take the ramp toward Harrisburg/Harrisburg International Airport."],
    "8": ["Turn left to take the PA 283 West ramp toward Harrisburg/Harrisburg International Airport."],
    "9": ["Turn right to take the Gettysburg Pike ramp."]
  }
},
```

- Please **DO** translate the JSON values. As needed, reorder the phrase words and tags - the tags must remain in the phrase. An example using the ramp instruction:

The **JSON values** (excluding the **phrase tags**) are highlighted in **green** below  
Please **translate** these items

```
"ramp": {
  "phrases": {
    "0": "Take the ramp on the <RELATIVE_DIRECTION>.",
    "1": "Take the <BRANCH_SIGN> ramp on the <RELATIVE_DIRECTION>.",
    "2": "Take the ramp on the <RELATIVE_DIRECTION> toward <TOWARD_SIGN>.",
    "3": "Take the <BRANCH_SIGN> ramp on the <RELATIVE_DIRECTION> toward <TOWARD_SIGN>.",
    "4": "Take the <NAME_SIGN> ramp on the <RELATIVE_DIRECTION>.",
    "5": "Turn <RELATIVE_DIRECTION> to take the ramp.",
    "6": "Turn <RELATIVE_DIRECTION> to take the <BRANCH_SIGN> ramp.",
    "7": "Turn <RELATIVE_DIRECTION> to take the ramp toward <TOWARD_SIGN>.",
    "8": "Turn <RELATIVE_DIRECTION> to take the <BRANCH_SIGN> ramp toward <TOWARD_SIGN>.",
    "9": "Turn <RELATIVE_DIRECTION> to take the <NAME_SIGN> ramp."
  },
  "relative_directions": ["left", "right"],
  "example_phrases": {
    "0": ["Take the ramp on the left.", "Take the ramp on the right."],
    "1": ["Take the I 95 ramp on the right."],
    "2": ["Take the ramp on the left toward JFK."],
    "3": ["Take the South Conduit Avenue ramp on the left toward JFK."],
    "4": ["Take the Gettysburg Pike ramp on the right."],
    "5": ["Turn left to take the ramp.", "Turn right to take the ramp."],
    "6": ["Turn left to take the PA 283 West ramp."],
    "7": ["Turn left to take the ramp toward Harrisburg/Harrisburg International Airport."],
    "8": ["Turn left to take the PA 283 West ramp toward Harrisburg/Harrisburg International Airport."],
    "9": ["Turn right to take the Gettysburg Pike ramp."]
  }
},
```



(Don't have a Github account? No problem! Follow the steps above to translate the routing instructions file in your language and email it to us at **hello@mapzen.com** (**mailto:hello@mapzen.com**))

If the task looks too daunting, just start a new language – someone in the community can help finish it!

## Footnotes

---

1. From <http://aboutworldlanguages.com/hindi> (<http://aboutworldlanguages.com/hindi>) ↗

· 18 November 2016 ·



**Ekta Daryanani**

Lead, Design + User Experience. Thinks: colors, people, community and city life. Does: mobile, maps, rock climbing and yoga.



**Varun Talwar**

Graphics Engineer. C++, Graphics and Physics Simulation enthusiast. Worked on Animation Movies.

---

© 2017 Mapzen

# Terrain Generalization

**tangram** (/tag/tangram) **terrain** (/tag/terrain) **demo** (/tag/demo)

“Generalization” is a term used in cartography to describe reducing the complexity of data while preserving its essential characteristics in a meaningful way. It’s not just simplification, but finding and revealing the most important details. It’s a useful skill when you have more information than you need, and that’s almost always the case.

Cartographers can learn to recognize significant details in mappable data, but teaching a computer this skill is trickier.

I’m interested in this problem lately as it relates to terrain. Mountains are a hobby of mine, and depicting them has made up a significant portion of my work at Mapzen (for more on this, see **Mapping Mountains** (<https://mapzen.com/blog/mapping-mountains/>)). Since that post, we’ve integrated terrain data in our **Walkabout and Tron house styles** (<https://mapzen.com/products/maps/>).

At the annual meeting of **NACIS** (<http://nacis.org>) (North American Cartographic Information Society) this year, the generalization of terrain was discussed in several sessions, giving me lots of ideas to explore. Included among these was a technique described by Daniel Huffman in his blog post **On Generalization Blending for Shaded Relief** (<https://somethingaboutmaps.wordpress.com/2011/10/18/on-generalization-blending-for-shaded-relief/>).

Generalization blending is a way to solve two problems at once – terrain data contains small details which aren’t necessary for understanding the size and shape of important features, but basic simplification methods such as blurring are applied everywhere indiscriminately.



*A displeasing, indiscriminately-blurred image.*

However, if you have some knowledge of the terrain’s bumpiness, you can use it to blur small details while letting bigger features show through in all their bumpy glory.

Following Daniel’s lead, I decided to try to implement a version of this technique in a realtime **Tangram** (<https://mapzen.com/products/tangram>) shader. To detect bumpiness, I used the **standard deviation** ([https://en.wikipedia.org/wiki/Standard\\_deviation](https://en.wikipedia.org/wiki/Standard_deviation)), which describes the range of variation in a collection of samples. Starting with our **tiled terrain normals** (<https://mapzen.com/documentation/terrain-tiles/>), we can sample a small area around each pixel to test how locally smooth or bumpy that area is. This will give us a “bumpiness” map with lines along ridges – the sharper the ridge, the brighter the line.

[Leaflet](#)

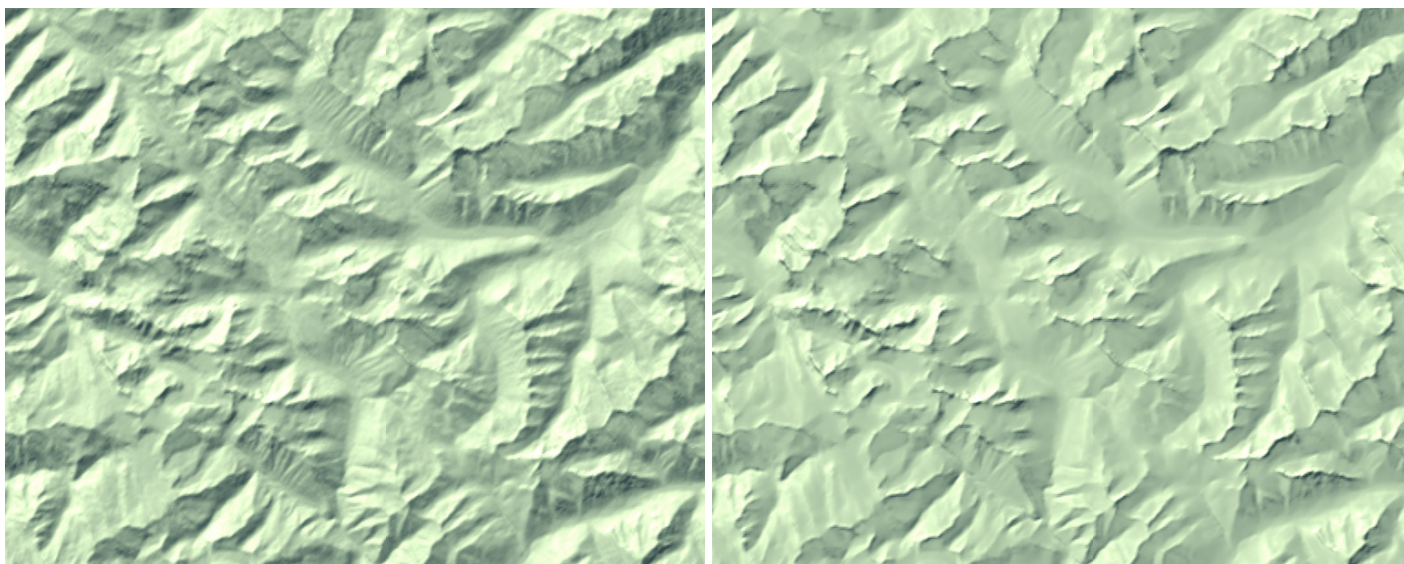
*An interactive bumpiness map of somewhere in British Columbia. [Open full screen ↗](#)*  
*(<http://tangrams.github.io/terrain-demos/?url=styles/green-stdev.yaml#10/57.0719/-126.2290>)*

This can be used as a mask, showing the original terrain only where the mask is bright, and fading to the blurred version everywhere else:

[Leaflet](#)

An interactive, selectively-blurred map. Open full screen ↗ (<http://tangrams.github.io/terrain-demos/?url=styles/green-selectiveblur.yaml#10/57.0719/-126.2290>)

Here's a side-by-side comparison:



*The original terrain, followed by selectively blurred terrain. I am pleased.*

Here's a **live, editable version of the demo** (<https://mapzen.com/tangram/play/?scene=https://raw.githubusercontent.com/tangrams/terrain-demos/master/styles/green-selectiveblur.yaml#10.1375/51.0141/-117.6778>), and here's a **link to the code** (<https://github.com/tangrams/terrain-demos/blob/master/styles/green-selectiveblur.yaml>) on github – fork away!

Further optimizations are possible – for example, **here's a version which should be faster** (<https://github.com/tangrams/terrain-demos/blob/master/styles/green-stdev-opt.yaml>) while giving slightly different results, and **here's a version which samples in screen space** (<https://github.com/tangrams/terrain-demos/blob/master/styles/green-stdev-adjusted.yaml>) to reduce artifacts at zoom levels (thanks to **Patricio Gonzalez Vivo** (<https://twitter.com/patriciogv>) and **Brett Camper** (<https://github.com/bcamper>) for shader help). And there are lots of other kinds of blurs and manipulations to try – for instance, the current shader works best out to about z8, but this range could be expanded by linking various shading parameters to the zoom level.

In this way, I believe terrain can be made legible at all scales, and then my life's work will be complete.

· 21 November 2016 ·



**Peter Richardson**

Peter vexes vertices, flusters fragments, and pesters pixels on Mapzen's graphics team.

---

© 2017 Mapzen

# Fixing Routing with Map Roulette

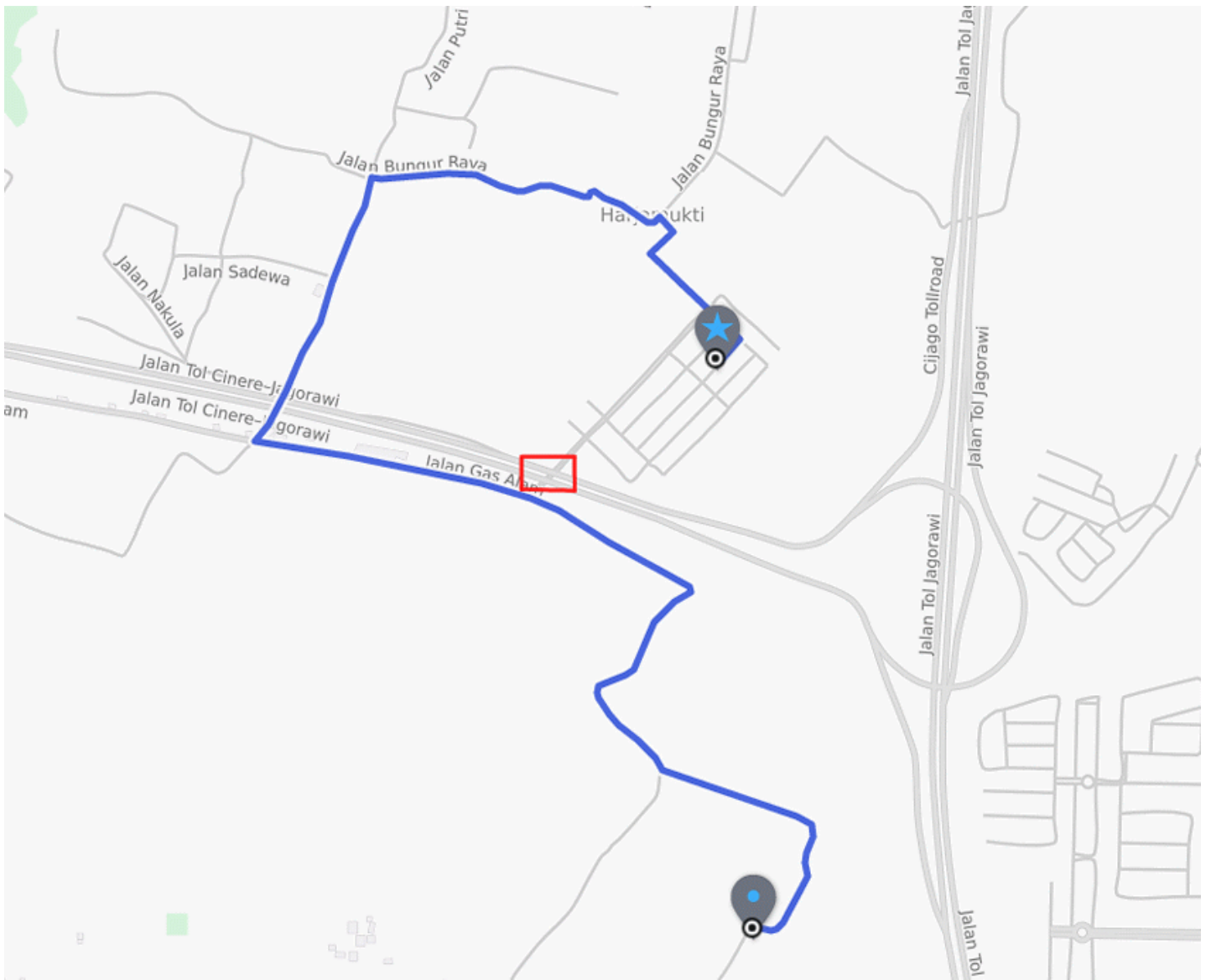
`routing (/tag/routing)` `osm (/tag/osm)` `engineering (/tag/engineering)`

Want to make OpenStreetMap even better? Try the **MapRoulette One-Way Challenge** (<http://maproulette.org/map/983>)!

Not so long ago the routing team noticed that the origin or destination of a route would occasionally be unreachable. After some investigation we realized that certain errors in OpenStreetMap could easily cause the routes to fail, especially when the direction of one-way roads was switched. To differentiate between routes that fail because two locations are disconnected in real life, and those that were doomed from the start by these wrong-way errors, we started trying to detect them.

To demonstrate the impact they can have, here is the same route before and after an error was fixed. The one-way road in the red box was reversed, forcing the route to go much farther than needed.





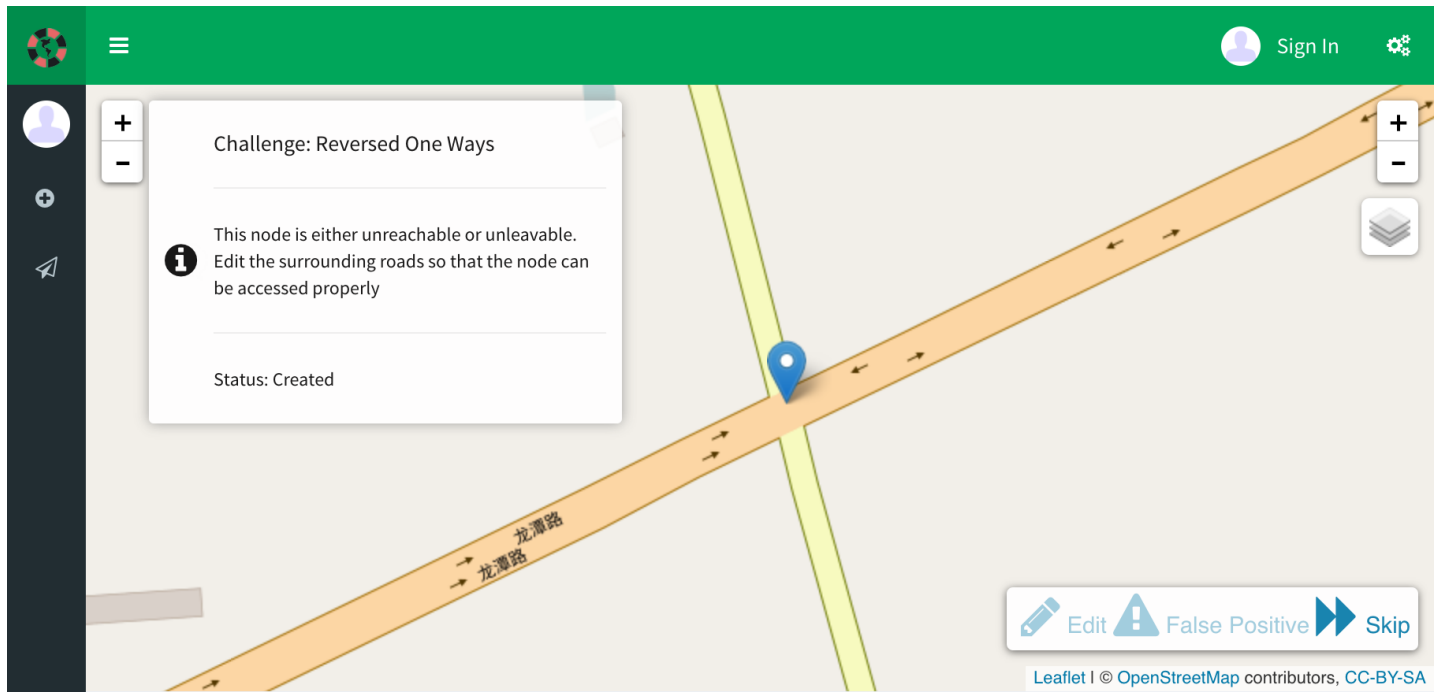
As we started working on the code to look for these errors, we had to figure out what to do with them. Do we manually edit them all ourselves? Do we ignore them and hope they get fixed down the road? There could be hundreds or thousands of these types of errors in OSM that could impact routing.

Enter **Map Roulette** (<http://maproulette.org/>), a platform that allows contributors to submit sets of errors so that the OSM community can fix them. We planned to submit the errors we were collecting to Map Roulette, but as we progressed, we realized that it was not enough to eliminate just one set of errors we had collected. Users are adding data to OpenStreetMap all of the time, and we needed a way to tag any errors made in the future.

We also thought that we could automate the process to minimize manual intervention. To start testing this we needed the help of **Martijn Van Exel** (<https://twitter.com/mvexel>), the creator of Map Roulette. He helped us get local instances of the Map Roulette server running as well as answering other questions we had along the way.

The automation process seemed like a good fit for **a bit of Python**

(<https://github.com/valhalla/mjolnir/tree/af5ec14df082c7bf7f5febeb63b91ef9be22c3aa/py>). After building a few tools to test the Map Roulette API with our collection of errors, we brought it all together into a system that checks the latest set of tiles without any manual intervention and automatically uploads new errors to Map Roulette. (Additionally, the tool can also detect previous errors that were incorrectly marked as fixed in Map Roulette.)



The result of all this work is a **general framework**

([https://github.com/valhalla/mjolnir/blob/af5ec14df082c7bf7f5febeb63b91ef9be22c3aa/py/MR\\_admin\\_tool.md](https://github.com/valhalla/mjolnir/blob/af5ec14df082c7bf7f5febeb63b91ef9be22c3aa/py/MR_admin_tool.md)) that allows for detection, submission, and eventual resolution of any kind of OSM errors. To start doing this for a new type of error, the only thing we would need to do would be to write a detection algorithm, create a GeoJSON representation of it, and add it to the tool's configuration file.

Currently, we only have one type of error that is being detected and automatically submitted, but look forward to seeing what other issues we can fix and how much we can help improve OSM. You can jump in and **fix these wrong-way errors right now in Map Roulette** (<http://maproulette.org/map/983>)!

· 21 November 2016 ·



**Kyle Hopkins**

Former Mapzen routing intern, friend of Python

© 2017 Mapzen

# Put A Label On It

[mapzen-js \(/tag/mapzen-js\)](#) [demo \(/tag/demo\)](#) [tutorial \(/tag/tutorial\)](#)

[tangram \(/tag/tangram\)](#) [make-your-own \(/tag/make-your-own\)](#)

Hello again!

We're back with the fourth installment of **Make Your Own** (<https://mapzen.com/tag/make-your-own/>) [ [Filters](#) ]. Today, we'll be focusing on how to add **labels** to your map using **mapzen.js** (<https://mapzen.com/documentation/mapzen-js/>) and the **Tangram** (<https://mapzen.com/documentation/tangram/>) scene file.

We are going to begin right where we left off last time, so if you haven't yet read the first few posts, I recommend starting there:

- **One Minute Map** (<https://mapzen.com/blog/one-minute-map/>)
- **Map Sandwich** (<https://mapzen.com/blog/map-sandwich/>)
- **Filters & Functions** (<https://mapzen.com/blog/filters-and-functions/>)

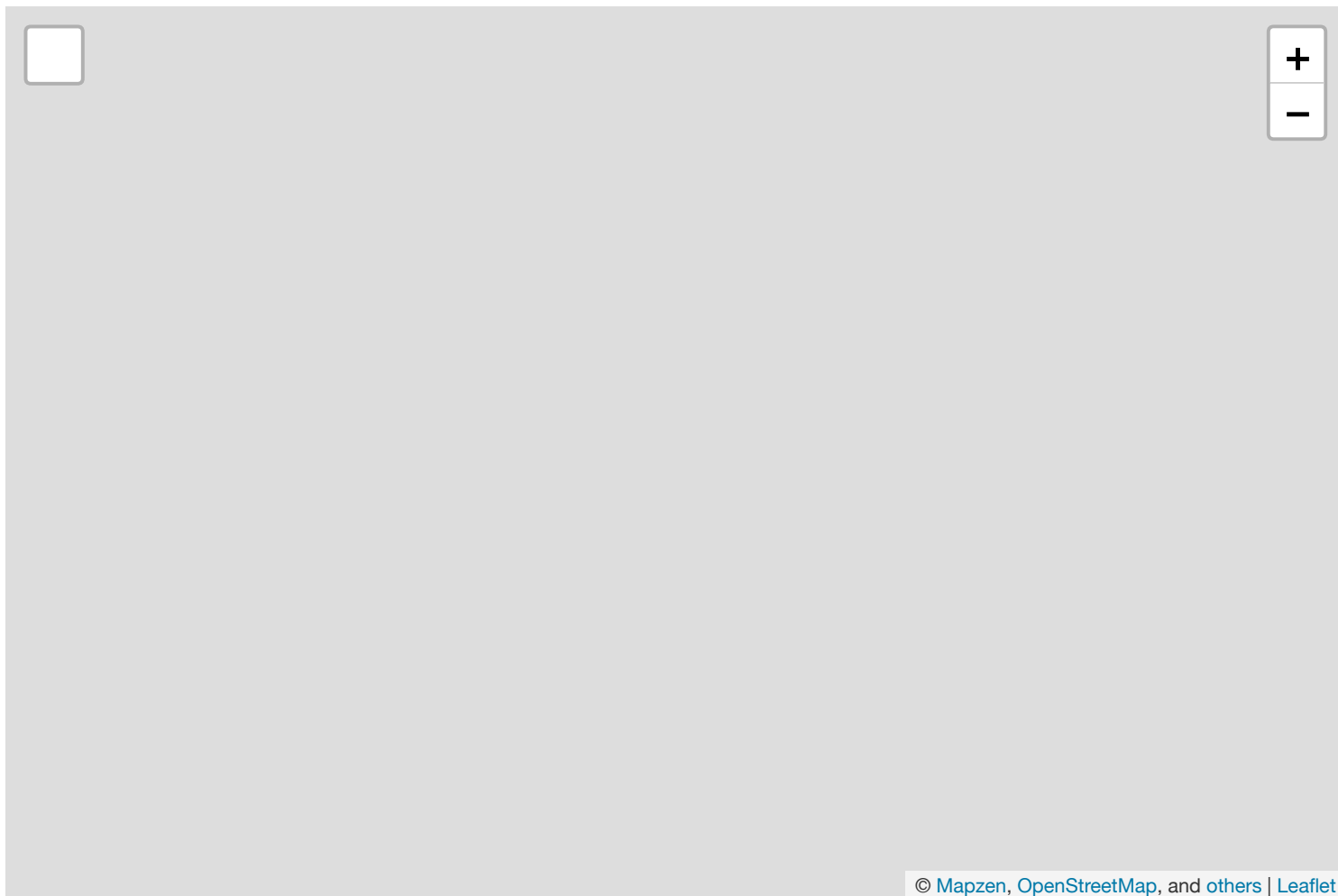
Once again, I'm going to assume you have a text editor and access to a web server. If you need either, please read our developer guide on **setting up a bare bones development environment** (<https://mapzen.com/documentation/guides/>) or use the gist → bl.ocks workflow described in **One Minute Map** (<https://mapzen.com/blog/one-minute-map/>).

Ok, let's get started!





Here's a reminder of what our geology map looks like from the **last post** (<https://mapzen.com/blog/filters-and-functions/>):



Looks pretty good, no?

*Buuuut....* it's maybe not the most helpful map I've ever made. (*What do these colors even mean?!*) Sure, we could slap on a legend, but I'd like to do one better. Let's add **labels** to the geologic units. That way we'll have a better idea of what we're looking at.

We'll start where we left off last time, with `index.html` :

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>San Juan Island Geology</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="https://mapzen.com/js/mapzen.css">
    <script src="https://mapzen.com/js/mapzen.min.js"></script>
    <style>
      #map {
        height: 100%;
        width: 100%;
        position: absolute;
      }
      html,body{margin: 0; padding: 0}
    </style>
  </head>
  <body>
    <div id="map"></div>
    <script>
      // Mapzen API key (replace key with your own)
      // To generate your own key, go to https://mapzen.com/developers/
      L.Mapzen.apiKey = 'mapzen-JA21Wes';

      var san_juan_island = [48.5326, -123.0879];

      var map = L.Mapzen.map('map', {
        center: san_juan_island,
        zoom: 12,
        scene: 'scene.yaml',
      });

      // Move zoom control to the top right corner of the map
      map.zoomControl.setPosition('topright');

      // Mapzen Search box (replace key with your own)
      // To generate your own key, go to https://mapzen.com/developers/
      var geocoder = L.Mapzen.geocoder('mapzen-JA21Wes');
      geocoder.addTo(map);
    </script>
  </body>
</html>

```

and scene.yaml :



```

import: https://mapzen.com/carto/walkabout-style/3/walkabout-style.zip

styles:
  _alpha_polygons:
    base: polygons
    blend: multiply
  _dashed_lines:
    base: lines
    dash: [3, 1]
    dash_background_color: rgb(149, 188, 141)

sources:
  _nps_boundary:
    type: GeoJSON
    url: https://gist.githubusercontent.com/rfriberg/684645c22f495b4a46f29fb312b6d268,

  _nps_geology:
    type: GeoJSON
    url: https://gist.githubusercontent.com/rfriberg/3c09fe3afd642224da7cd70aff1c1e70,

layers:
  _national_park:
    data: { source: _nps_boundary }
    draw:
      _dashed_lines:
        width: [[8, 0.5px], [18, 5px]]
        color: '#518946'
        order: global.sdk_order_over_everything_but_text_1

  _geology:
    data: { source: _nps_geology }
    filter:
      all:
        - { $zoom: { min: 10 } }
        - not: { GLG_SYM: water }
    draw:
      _alpha_polygons:
        order: global.sdk_order_over_everything_but_text_0
        color: |
          function() {
            // Note: this is a block of JavaScript so we can use JS comment s
            var category = feature.GLG_SYM;
            var color = category == 'Qa'      ? '#FFF79A' :
                          category == 'Qb'      ? '#FFF46E' :
                          category == 'Qd'      ? '#fff377' :
                          category == 'Qf'      ? '#dddddd' :
                          category == 'Qp'      ? '#EAC88D' :
                          category == 'Qgdm'    ? '#FCBB62' :

```

```

category == 'Qgdm(es)' ? '#FEE9BB' :
category == 'Qgdm(e)' ? '#E8A121' :
category == 'Qgom(e)' ? '#EAB564' :
category == 'Qgom' ? '#FECE7A' :
category == 'Qgd' ? '#FEDDA3' :
category == 'Qgt' ? '#FCBB62' :
category == 'KJmm(c)' ? '#86C879' :
category == 'KJm(ll)' ? '#9FD08A' :
category == 'JTRmc(o)' ? '#27BB9D' :
category == 'TRn' ? '#ED028C' :
category == 'TRPMv' ? '#F172AC' :
category == 'TRPv' ? '#F499C2' :
category == 'PDmt' ? '#40C7F4' :
category == 'pPsh' ? '#9BA5BE' :
category == 'pDi' ? '#848FC7' :
category == 'pDit(t)' ? '#B28ABF' :
'#000';

return color;
}

```

*UPDATE March 1, 2017: A Mapzen developer API key is now required for mapzen.js. We've updated the Make Your Own series to include a demo key. Generate your own free API key at <https://mapzen.com/developers/> (<https://mapzen.com/developers/>).*

## Layer or Sublayer

As you can see above, our scene file has two **layers**

(<https://mapzen.com/documentation/tangram/layers/>): **\_national\_park** (**San Juan National Historic Park boundaries** (<https://catalog.data.gov/dataset/national-park-boundariesf0a4c>)) and **\_geology** (**San Juan geologic units** (<https://catalog.data.gov/dataset/digital-geologic-map-of-san-juan-island-national-historical-park-and-vicinity-washington-nps-gr>)).

One option for drawing our labels would be to add a third layer to our scene file. Something along the lines of:

```

_geology_labels:
  data: { source: _nps_geology }
  draw: ...

```

However, because we are using the same data source as our `_geology` layer, we could turn our layer into a **sublayer** (<https://mapzen.com/documentation/tangram/layers/#sublayer-name>) of `_geology`. The sublayer will **inherit** (<https://mapzen.com/documentation/tangram/Filters-Overview/#inheritance>) both the data source *and* the filter that we set up for our `_geology` layer, which will effectively reduce the number of parameters we need to set. (**#winning** ([http://www.reactiongifs.com/wp-content/uploads/2013/06/Sean\\_connery\\_raction.gif](http://www.reactiongifs.com/wp-content/uploads/2013/06/Sean_connery_raction.gif)))

Ok, so let's call our sublayer "`_geology_labels`" and insert it within the `_geology` layer, just below the draw block. I'll add a comment to make it a little easier to follow:

```
_geology:
  data: ...
  filter: ...
  draw: ...

# Labels sublayer
_geology_labels: ...
```

## Labels

To draw our labels, we'll be using one of Tangram's built-in **draw styles** (<https://mapzen.com/documentation/tangram/Styles-Overview/#text>) called `text`. The `text` draw style ([https://mapzen.com/documentation/tangram/Styles-Overview/#text\\_1](https://mapzen.com/documentation/tangram/Styles-Overview/#text_1)) will draw a text label at a given point, depending on the type of data that is provided. For point data, the label will be drawn at the point. For lines, the label will be drawn along the line. And for polygons (like our data), text will be drawn within the polygon at regular spacing *or* at the polygon's centroid. (*More on that shortly.*)

To our scene file, let's add a simple, black label:

```

_geology:
  data: ...
  filter: ...
  draw: ...

# Labels sublayer
_geology_labels:
  draw:
    text:
      text_source: GLG_SYM
      font:
        fill: black
        size: 12px

```

We're using the **text\_source**

([https://mapzen.com/documentation/tangram/draw/#text\\_source](https://mapzen.com/documentation/tangram/draw/#text_source)) parameter to set the source of the label text. This could be a function (that returns a string) or the name of a feature property. If you recall from our last post in the series (**Filters and Functions** (<https://mapzen.com/blog/filters-and-functions/>)), each of our GeoJSON features comes with a set of feature properties:

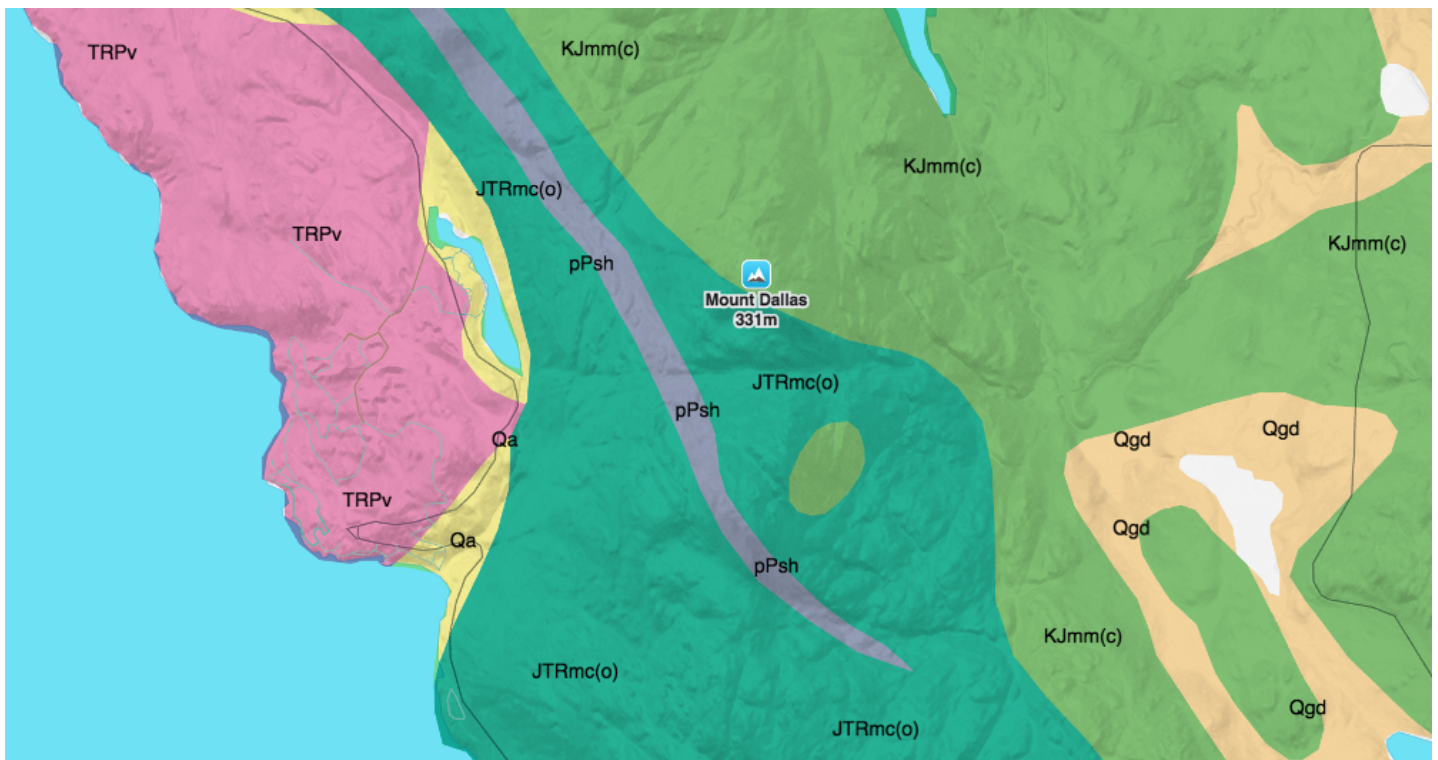
```

{
  "type": "Feature",
  "properties": {
    "FUID": 1,
    "GLG_SYM": "KJmm(c)",
    "SRC_SYM": "KJm(c)",
    "SORT_NO": 13.0,
    "NOTES": "NA",
    "GMAP_ID": 74832,
    "HELP_ID": "KJmm(c)",
    "SHAPE_Leng": 137.95199379300001,
    "SHAPE_Area": 954.47301117400002
  },
  "geometry": ...
}

```

These properties are accessible to a few different blocks, including **filters** (<https://mapzen.com/documentation/tangram/Filters-Overview/#feature-properties>) and **text\_source** ([https://mapzen.com/documentation/tangram/draw/#text\\_source](https://mapzen.com/documentation/tangram/draw/#text_source)). In this case, we're telling Tangram to show the value stored in the feature property "GLG\_SYM" as the text for our label.

If you've updated your map, it should look something like this:



There are several **text parameters** (<https://mapzen.com/documentation/tangram/Styles-Overview/#text-parameters>) we can use to modify our labels, but the only parameter that needs to be set is `font`. So far, we've set our label size and color, but let's take a look at some of the other **font parameters** (<https://mapzen.com/documentation/tangram/draw/#font-parameters>) we can use to improve the look of our labels.

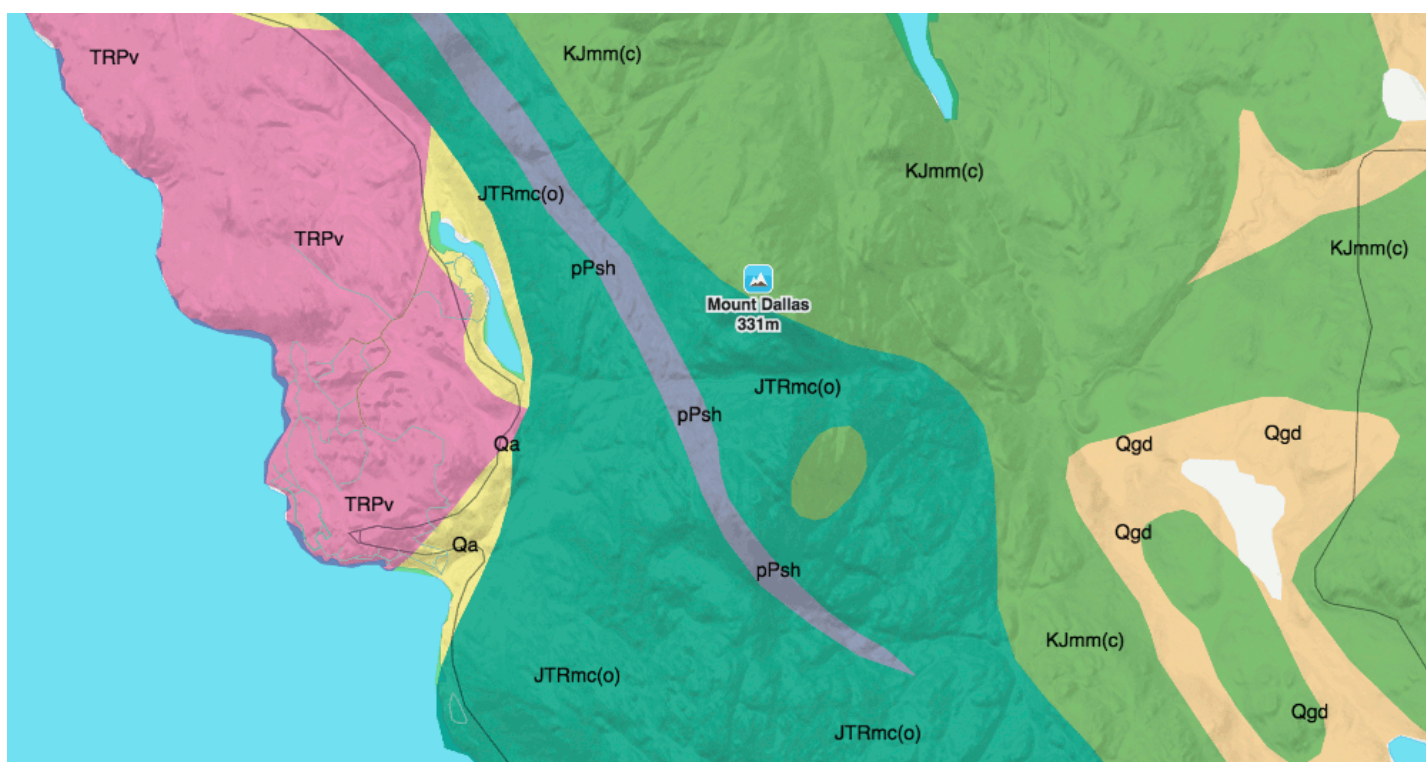
Let's change our **font weight** (<https://mapzen.com/documentation/tangram/draw/#weight>) to `bold` and update the color of our text to something that blends a little better: `rgba(130, 84, 41, 0.9)`. We'll also change our fixed size to a dynamic size using **stops** (<https://mapzen.com/documentation/tangram/yaml/#stops>) (which you may remember from our **Map Sandwich** post (<https://mapzen.com/blog/map-sandwich/>)):

```
text:
  text_source: GLG_SYM
  font:
    fill: rgba(130, 84, 41, 0.9)
    size: [[13, 10px], [20, 24px]]
    weight: bold
```

Ok, that's better. Let's do one more thing to make these labels pop a little more. Let's add a **stroke** (<https://mapzen.com/documentation/tangram/draw/#stroke>) to the font. Stroke might be better described as an outline or halo around the text and takes only two properties:

```
weight: bold
stroke:
  color: rgba(242, 218, 193, 0.25)
  width: 3
```

I'm using a low opacity value in my rgba, so as to not make the labels too distracting. What do you think?



Ok, I'll be honest. There is one thing that's still bothering me.

There are just *so many labels*. We don't need so many duplicate labels. Who does that help?

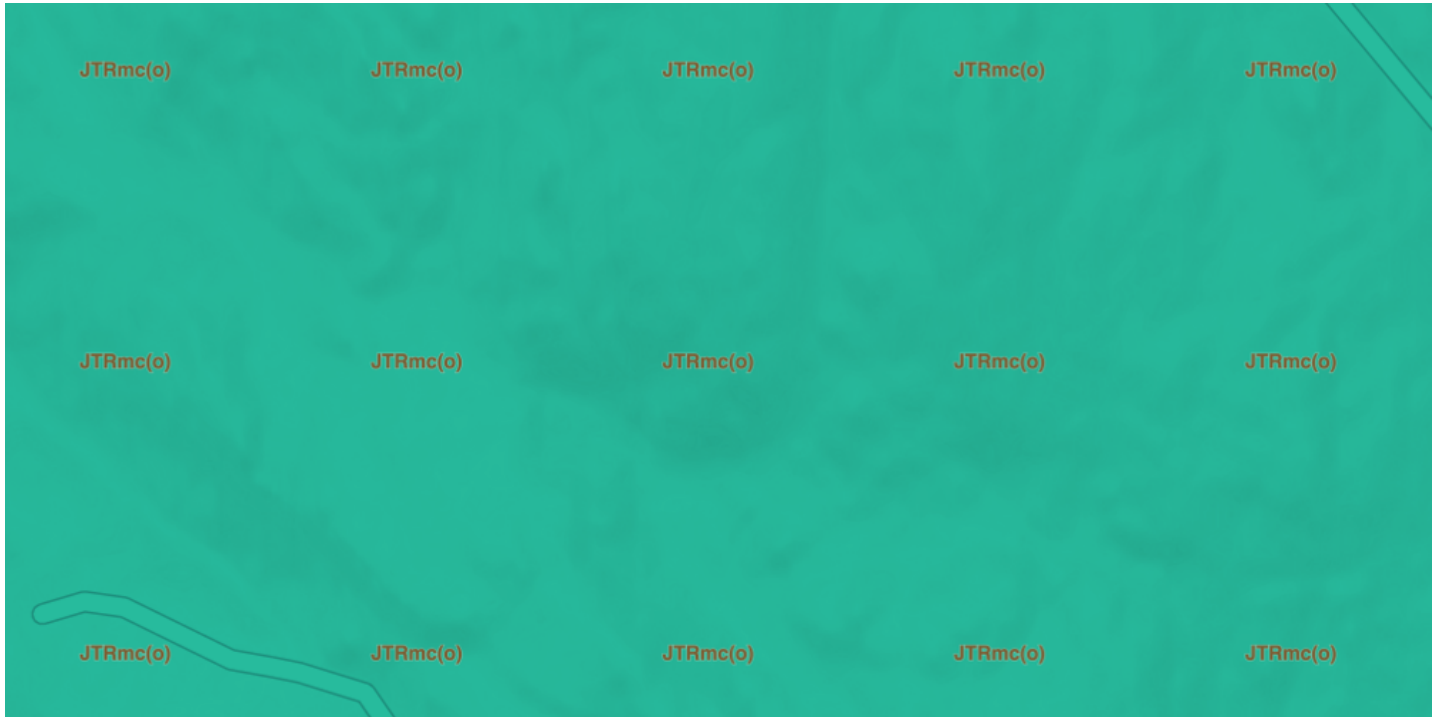
On first look, it seems like **repeat\_distance** ([https://mapzen.com/documentation/tangram/draw/#repeat\\_distance](https://mapzen.com/documentation/tangram/draw/#repeat_distance)) might be our answer. The `repeat_distance` parameter specifies the minimum distance between labels. That's great! We can just bump up that number and away we go!

Well... not so fast.

There's one thing we're forgetting.

Tangram displays data in tiles. And each tile interprets `repeat_distance` separately. So, while a label may not repeat *within* a tile, it can still repeat *across* tiles. This can result in multiple labels per feature, spread across different tiles.

If we zoom in on a single geologic unit, this effect becomes quite obvious:



I know I've said this before, but... we can fix that!

**Enter: the *generate\_label\_centroids* parameter.**

There is an optional parameter that we can pass to our data source, called

**`generate_label_centroids`**

([https://mapzen.com/documentation/tangram/sources/#generate\\_label\\_centroids](https://mapzen.com/documentation/tangram/sources/#generate_label_centroids)). This will create a label point at the centroid of each of our polygon features. It will also assign a `{label_placement: true}` ([https://mapzen.com/documentation/tangram/Filters-Overview/#label\\_placement](https://mapzen.com/documentation/tangram/Filters-Overview/#label_placement)) property to each of the centroid labels, which will allow us to filter the labels to show only those at the polygon centroid.

Did that make sense? Let's take a look at it in action.

In our `sources` block, we'll update our `_nps_geology` layer to include this new parameter:



```
_nps_geology:
  type: GeoJSON
  url: https://gist.githubusercontent.com/rfriberg/3c09fe3afd642224da7cd70aff1c1e70,
  generate_label_centroids: true
```

Then we'll add a filter to `_geology_labels` to ensure we're only displaying the centroid labels:

```
# Labels sublayer
_geology_labels:
  filter: { label_placement: true }
  draw:...
```

At this point, your `geology_labels` sublayer should look something like this:

```
# Labels sublayer
_geology_labels:
  filter: { label_placement: true }
  draw:
    text:
      text_source: GLG_SYM
      font:
        fill: rgba(130, 84, 41, 0.9)
        size: [[13, 10px], [20, 24px]]
        weight: bold
        stroke:
          color: rgba(242, 218, 193, 0.3)
          width: 3
```

We should note that `_geology_labels` is still following the rules of its parent layer's filter; it has simply added a second set of filtering rules. We can further refine this filter, by displaying our labels at zoom level 13 and above:

```
# Labels sublayer
_geology_labels:
  filter: { label_placement: true, $zoom: { min: 13 } }
```

By the way, the format I used above ( `{ label_placement: true, $zoom: { min: 13 } }` ) is a **shortcut** (<https://mapzen.com/documentation/tangram/Filters-Overview/#lists-imply-any-mappings-imply-all>) for mapping multiple filters using the `all` filter. This is equivalent to:

```
# Labels sublayer
_geology_labels:
  filter:
    all:
      - label_placement: true
      - $zoom: { min: 13 }
```

Phew! Labels. Amiright?

This is usually the point where I am *extremely grateful* that I can build off of gorgeous, **existing** cartography (like the **Mapzen basemaps** (<https://mapzen.com/documentation/cartography/styles/>) \*cough\* \*cough\*), and don't have to worry about things like **highway shields** (<https://mapzen.com/blog/shields/>) or **water transformations at different zoom levels** (<https://mapzen.com/blog/tron-v2-visual-scale/>).

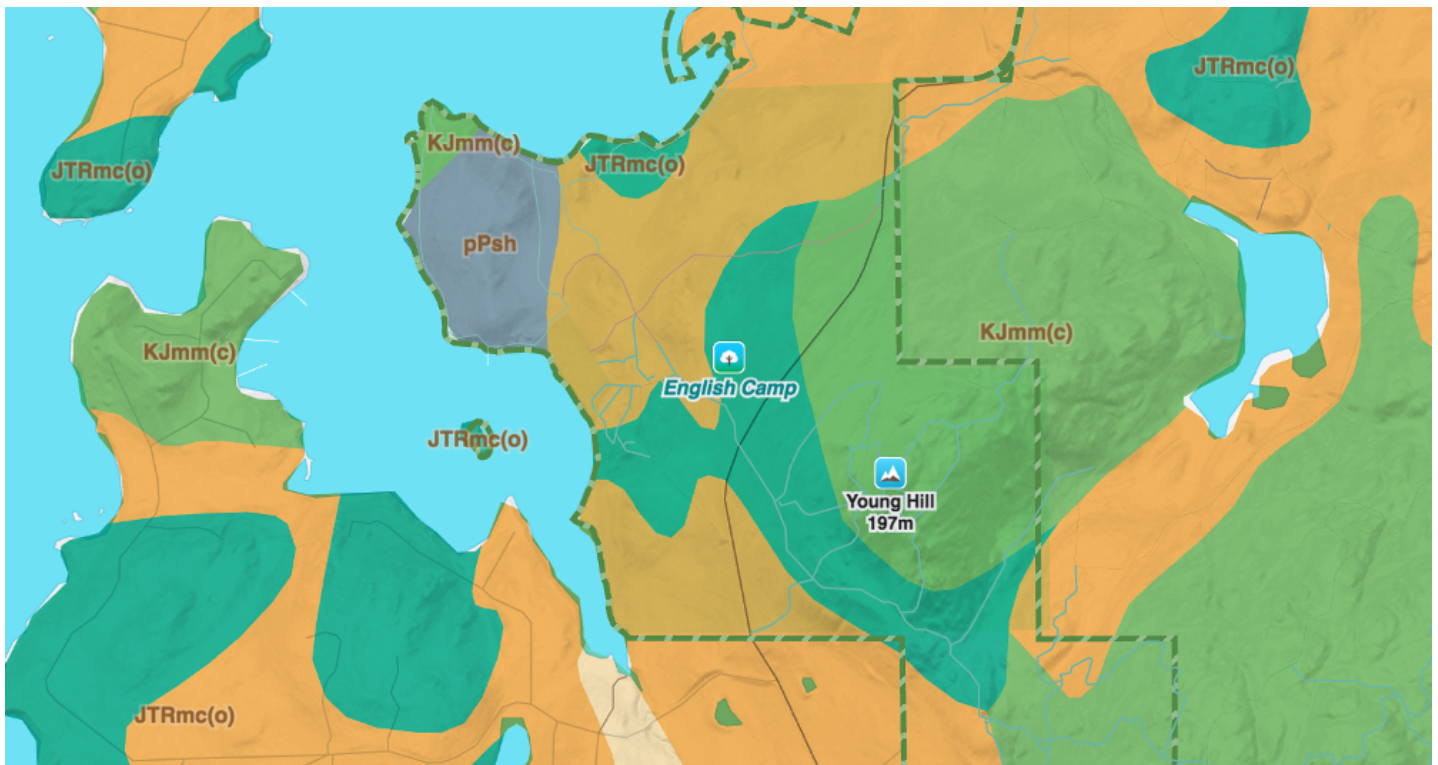
*I sense a segue coming...*

Though there are times when that gorgeous cartography *might* get in the way of our map's primary focus.

*Ah, there it is.*

## Overriding basemap features

Perhaps it's better if I show you what I mean. Let's take a closer look at English Camp on our map:



Have you noticed something a little *off* with that orange color within our national park boundary? What's happening is that orange (as well as the surrounding colors, though less noticeably so) is blending with **Walkabout**

(<https://mapzen.com/documentation/cartography/styles/#walkabout>)'s `landuse` layer, which is typically a nice, soothing green. In our case, however, the colors blend to create a somewhat distracting region of muted colors.

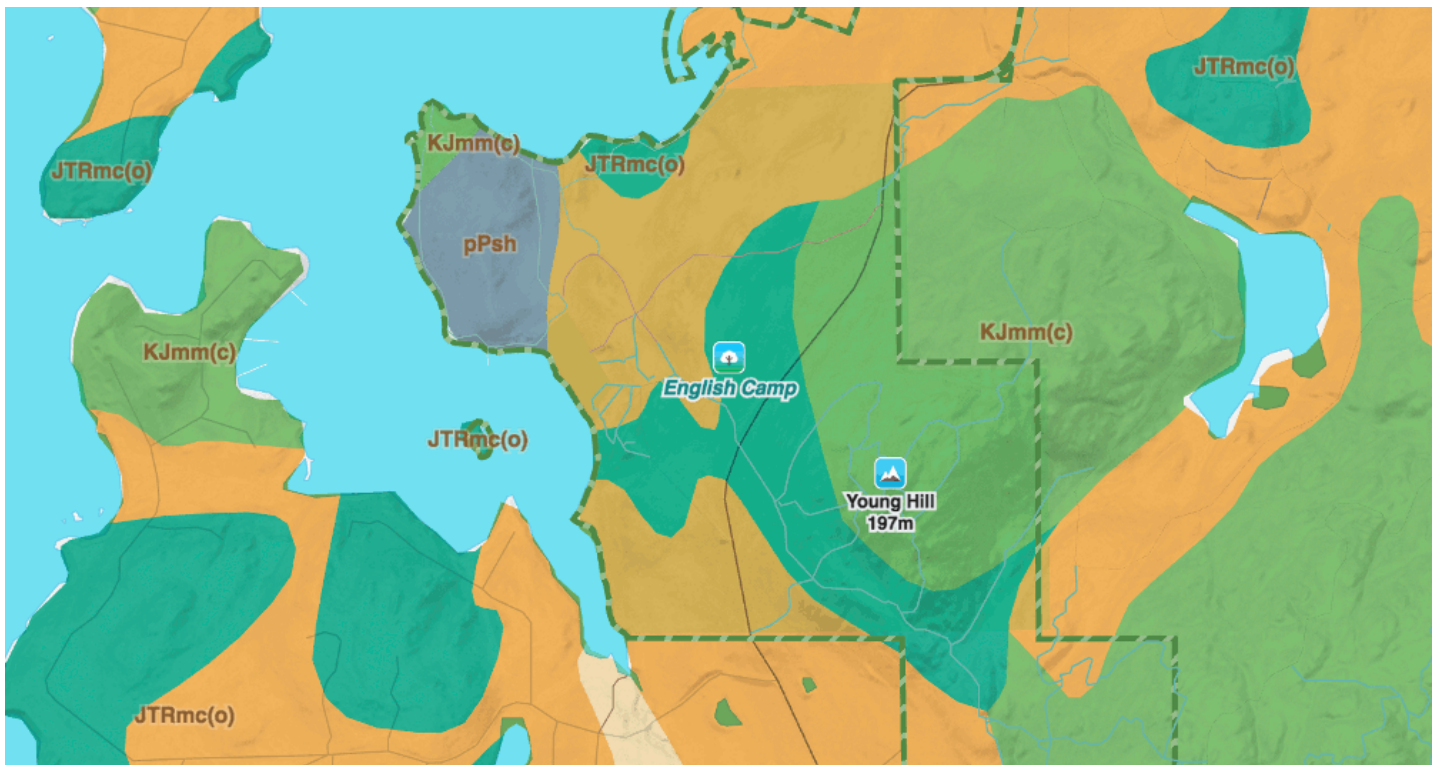
Maybe it's not a big deal. Maybe I'm being picky. Let's fix it just the same.

The easiest way to fix this is to override Walkabout's `landuse` layer and simply set its visibility to `false`. At the bottom of your scene file, add the following layer:

```
landuse:  
  visible: false
```

Because this layer has already been defined in an imported scene file, we don't need to set any other parameters. We simply pass in the parameter we want to override.

Easy!



Well, *easy* if you happen to share an office with the architects of Walkabout. Admittedly, it's *much less easy* if you need to sift through the **Walkabout yaml file** (<https://mapzen.com/cartto/walkabout-style/2/walkabout-style.yaml>) to figure out layer names and settings.

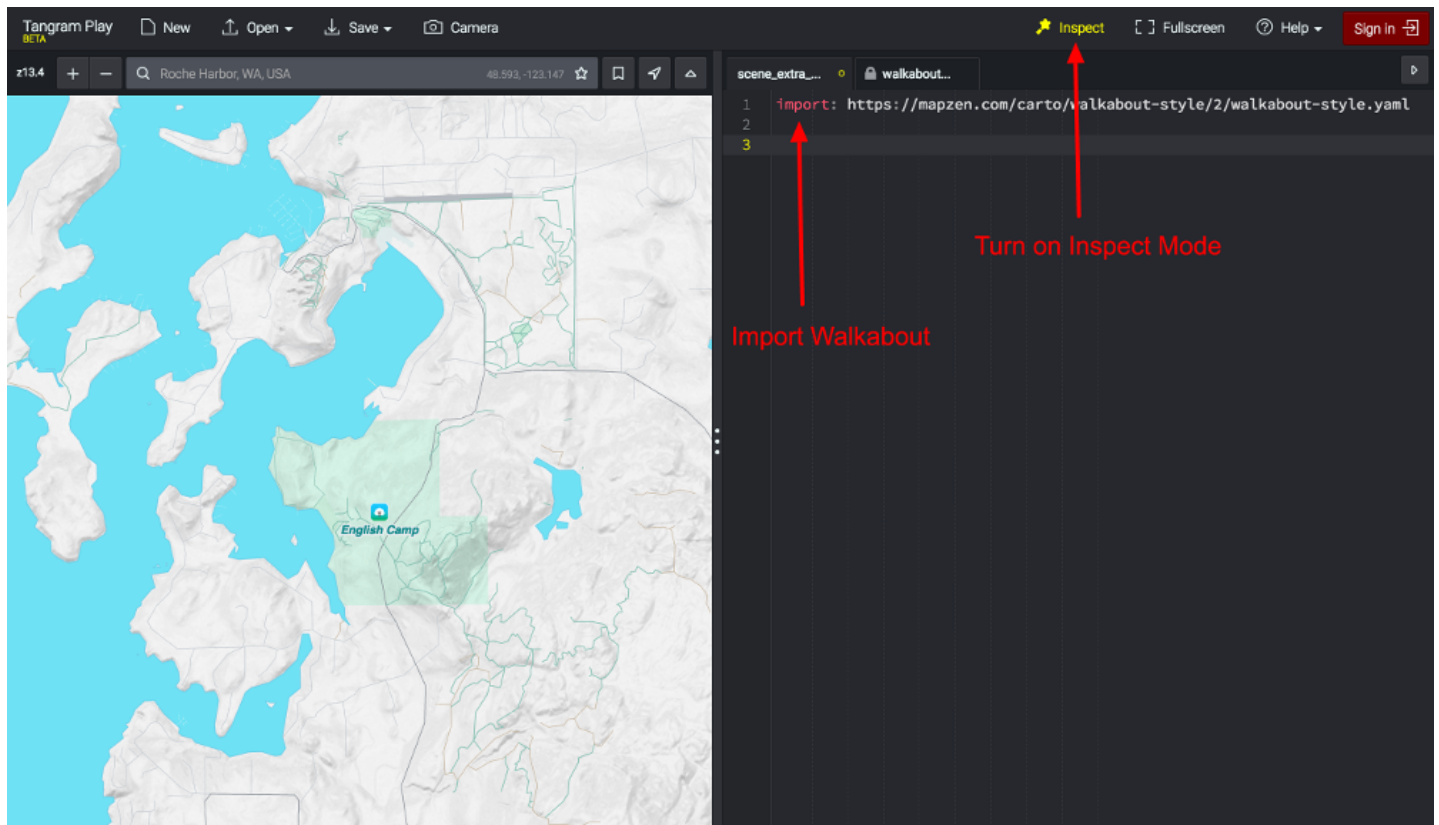
To make this is a little less painful, we'll use **Tangram Play** (<https://mapzen.com/tangram/play/>), Mapzen's live Tangram style editor.

In your browser, go to <https://mapzen.com/tangram/play/#13.4608/48.5872/-123.1450> (<https://mapzen.com/tangram/play/#13.4608/48.5872/-123.1450>) (this link will position you right over English Camp on San Juan Island).

In the `default.yaml` file on the right, delete everything and replace with a single line:

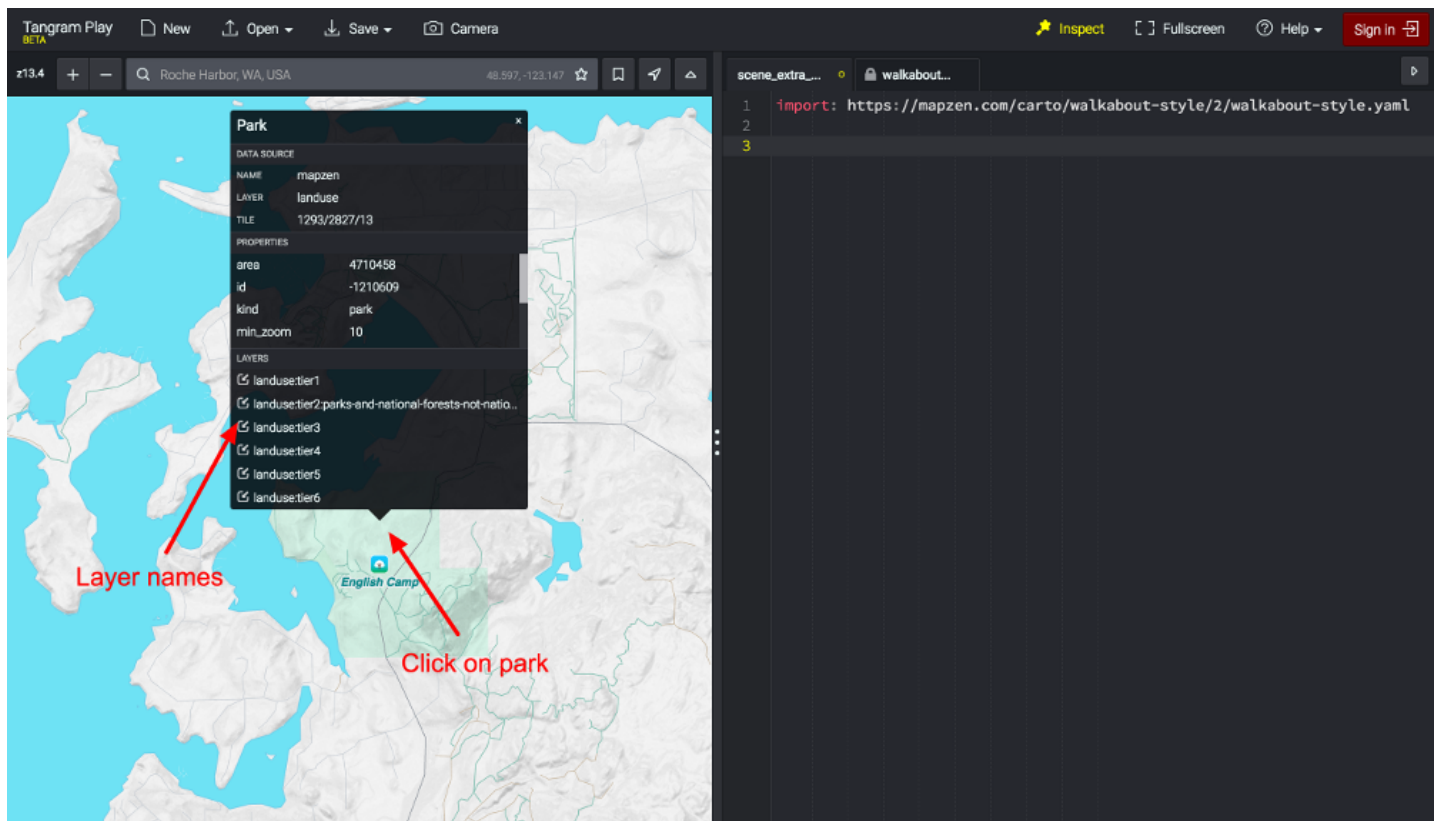
```
import: https://mapzen.com/cartto/walkabout-style/3/walkabout-style.zip
```

Then click the **Inspect** button in the top right corner.



**Inspect mode** will allow you to move your cursor across the map and get information about each layer, including the layer name(s) that Walkabout uses for styling.

Go ahead and click on the green national park polygon:

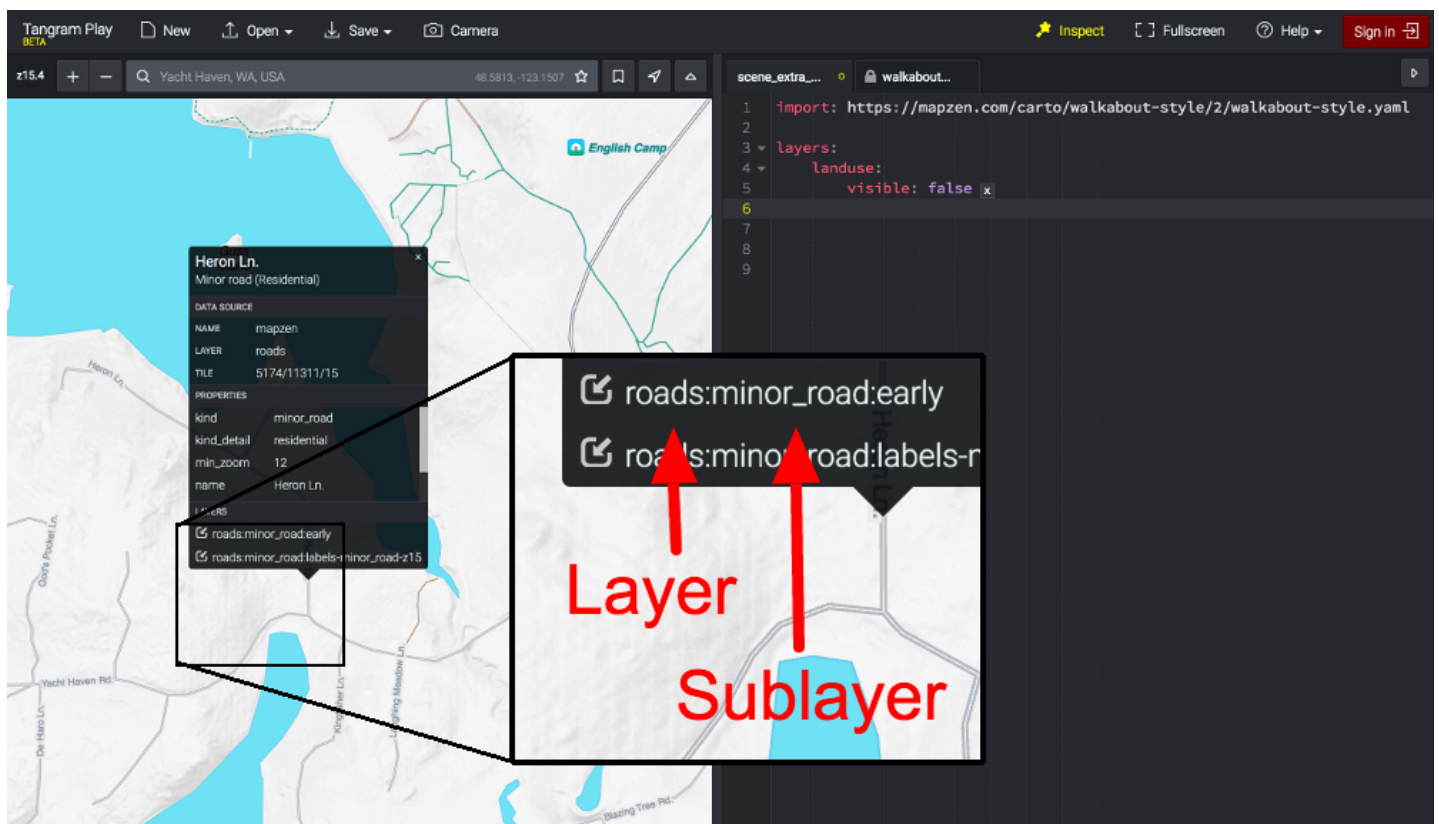


As you'll see, it's part of a layer named "landuse". In the `default.yaml` file, add the following just beneath the `import` line:

```
layers:
  landuse:
    visible: false
```

You should no longer see the green of the national park. We have effectively turned off the entire `landuse` layer.

We can do the same with sublayers. Just south of the park are several small roads, which I'd like to hide on our map. If you click on one of the minor roads, you'll see the layer information:



At the bottom, you'll see that `minor_road` is a sublayer of `roads`. We don't want to turn off all roads, just the minor roads, so we'll want to target just this sublayer.

In the `default.yaml` file, add another layer reference:

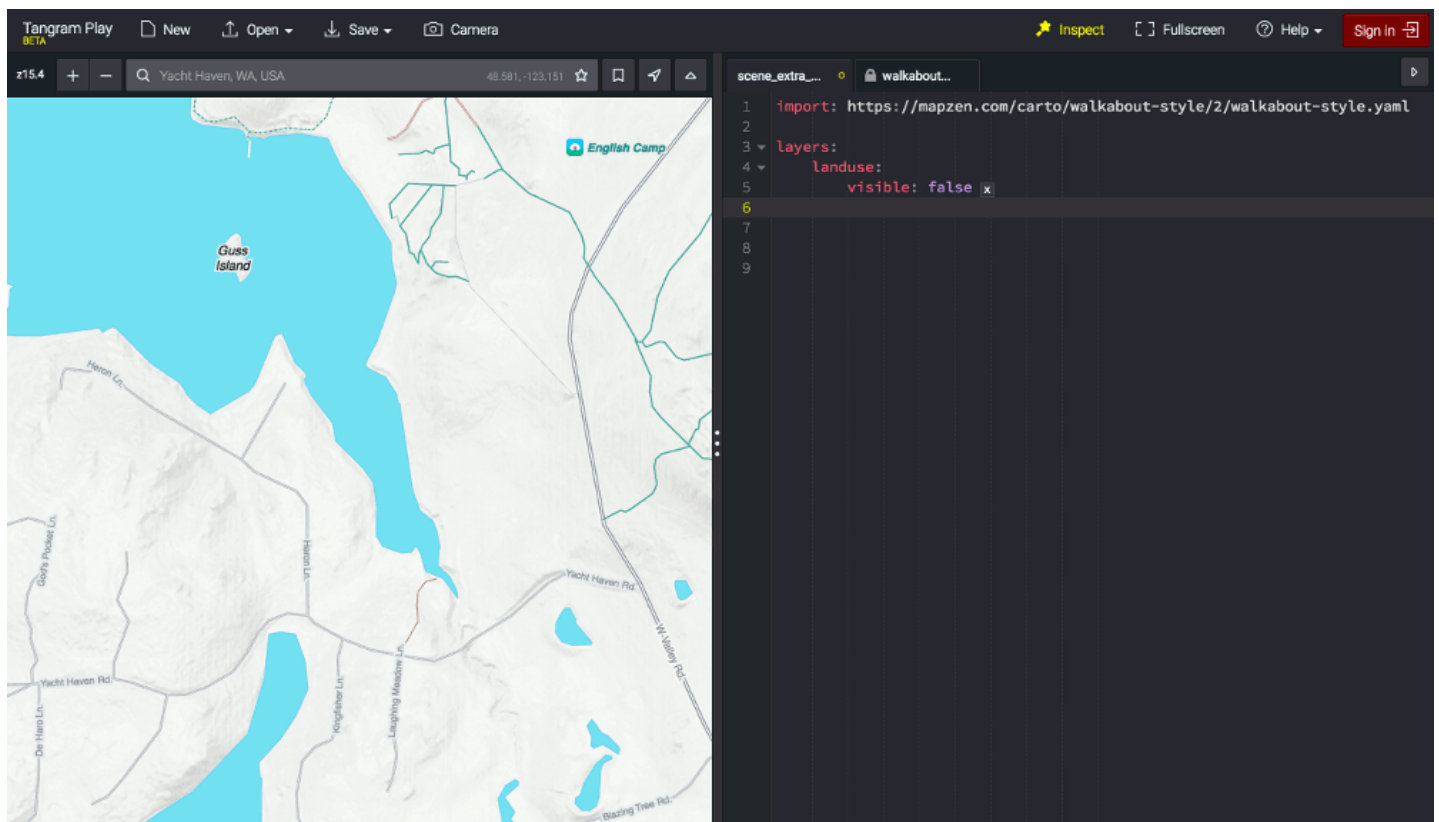


```
roads:
  minor_road:
    visible: false
```

As expected, this turns off *only* the `minor_road` sublayer. This method doesn't just work for the visibility parameter. If we wanted, we could make other changes to the existing layer, including changing the road color, width, and outline:

```
roads:
  minor_road:
    draw:
      lines:
        color: red
        width: 8px
        outline:
          color: darkred
          width: 1px
```

It's not quite the cartographic marvel I was going for, but you get the idea.

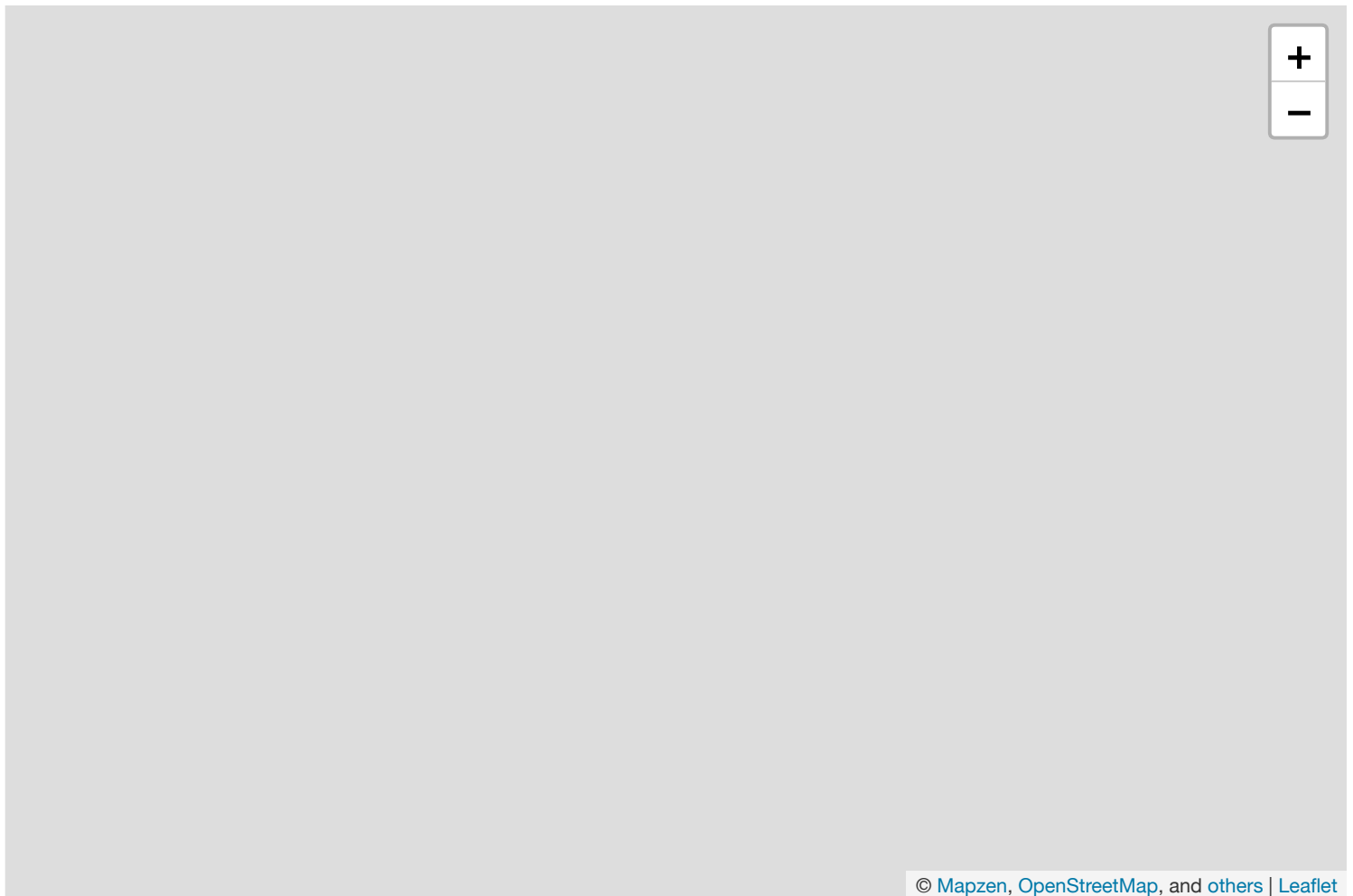




Once we've figured out which layers we want to turn off (or modify) using Tangram Play, we can then copy and paste those layers back into our local `scene.yaml` file:

```
landuse:
  visible: false
roads:
  minor_road:
    visible: false
```

And we're done!



The full code for this exercise can be seen on **bl.ocks.org** (<https://bl.ocks.org/rfriberg/5eebfa479ede198c9ba5a8545b838620>). You can also view and modify the final scene file using **Tangram Play** ([https://mapzen.com/tangram/play/?scene=https%3A%2F%2Fs3.amazonaws.com%2Fmapzen-assets%2Fimages%2Fgeology-labels%2Fscene\\_demo\\_final.yaml#12.1816/48.5378/-123.0883](https://mapzen.com/tangram/play/?scene=https%3A%2F%2Fs3.amazonaws.com%2Fmapzen-assets%2Fimages%2Fgeology-labels%2Fscene_demo_final.yaml#12.1816/48.5378/-123.0883)).

Thanks for coming along on another whirlwind tour of **mapzen.js** (<https://mapzen.com/documentation/mapzen-js/>) and the **Tangram** (<https://mapzen.com/documentation/tangram/>) scene file. Stay tuned for the next **Make Your Own** (<https://mapzen.com/tag/make-your-own/>) [ [Filters](#) ] post when we'll talk about ways we can add interactivity to our map. It's gonna be a good one!

If you have questions or want to show off something you've made with Mapzen, **drop us a line** (<mailto:hello@mapzen.com>). We love to hear from you!

~~~

Check out additional tutorials from the **Make Your Own** (<https://mapzen.com/tag/make-your-own/>) series:

- **One Minute Map** (<https://mapzen.com/blog/one-minute-map/>)
- **Map Sandwich** (<https://mapzen.com/blog/map-sandwich/>)
- **Filters & Functions** (<https://mapzen.com/blog/filters-and-functions/>)
- **Put A Label On It** (<https://mapzen.com/blog/tangram-labels/>)
- **Interactive Mapping with Tangram** ([https://mapzen.com/blog/tangram-
interactivity/](https://mapzen.com/blog/tangram-interactivity/))
- **Lots of Dots** (<https://mapzen.com/blog/lots-of-dots/>)
- **Customize Your Search** (<https://mapzen.com/blog/customize-your-search/>)

· 29 November 2016 ·



Rhonda Friberg

Rhonda is a javascripter who makes maps, trouble, and the occasional pie.

Unearthing CircleCI artifacts

engineering (/tag/engineering)

We're big fans of **CircleCI** (circleci.com) at Mapzen – it's our mane tool for automated software testing to make sure new code doesn't run a-foal. The repository everyone at Mapzen touches is our website, which used to stirrup a lot of emotions when we wanted to preview changes and share it with teams. **Mike Migurski** (<https://twitter.com/michalmigurski>) built an open-source tool called **Precog** (<https://precog.mapzen.com/>) that uses CircleCI artifacts to help preview our Jekyll-generated websites. Our favorite part of Precog, is the loading preview that reminds us to 'hold our horses'.

Hold Your Horses

This **preview is still being built**. It will load when it is ready.

CircleCI thought it was pretty cool and **Alek Sharma talked to Mike about it** (<https://circleci.com/blog/how-mapzen-uses-circleci-artifacts-for-static-site-preview-generation/>).

MM: Yeah, it's a Python-based Flask app. It doesn't even really need to know anything in particular ahead of time. You can basically feed it any path. It's: hostname/account/, and with that information, it'll go talk to the CircleCI API and say, "Hi, do you know anything about this repo?" Then, it'll talk to GitHub and get more information about the repo, and if it finds artifacts, it will show them.

AS: And then you can get more specific with commit SHAs on the actual branches.

MM: Exactly. You can look at different branches or individual commits. If you have a private repo, you can do a GitHub OAuth in order to show private stuff. For a long time, actually, private was the only thing we did, but then we started to see that there was a use-case for public repos inside Mapzen, so we set it up for both private and public repos.

AS: We've touched on this, but let's talk about how Mapzen uses Precog now.

MM: Basically, any of our repositories that generates a public static file has Precog attached to it. Just four off the top of my head: our website runs on Jekyll; that's got static output, so we use Precog. Our documentation system uses a Python library called Mkdocs. It's kind of "Jekyll for documentation", basically. Same deal: it outputs the artifacts, so we use Precog to preview it. Our Mapzen.js library has an NPM-based build process, so that outputs static files, so we use Precog with that. Then, our style guide, which is our front-end pattern library, uses Jekyll, so same deal. All of those things get Precog URLs; you can look at old versions and future branches of them.

AS: So, openness is a big deal. Has it always been that way?

MM: It's part of Mapzen's mission, and it was always founded to be that way... [with open source], you can support uses that are totally outside those predicted areas.

If you use Github branches and CircleCI, horse around with **Precog** (<https://precog.mapzen.com/>)! It's open and available for anyone to use. 🐎

Preview GIF: Galloping Horse zoopraxiscope by Eadweard Muybridge via Wikipedia (https://en.m.wikipedia.org/wiki/Eadweard_Muybridge)

· 02 December 2016 ·



Mapzen

Opening maps with open source and open data.

© 2017 Mapzen

Do Less, Get More!

geocoding (/tag/geocoding)

Wouldn't it be lovely if we could do a little bit less but in doing so, improved things? We're excited to report that we have been able to make this paradox a reality! We've finally added what we are calling *structured geocoding* to our open source geocoding engine, Pelias, and thereby Mapzen Search.

In the past we've asked our users to do additional work when geocoding tabular address data that was split into constituent parts. Because we only supported a single string query parameter, our users were forced to concatenate the address parts into a single string in order to geocode that address.

Not only did this make things harder for the user, it in fact made things harder for the geocoding engine as well. The engine was consequently tasked with accurately breaking up the single query string back into the very parts the user glued together in the first place. As you can imagine, this process leaves room for misunderstandings. It was often the case that result accuracy suffered due to this unnecessary input processing.

Enter **Structured Geocoding!**

Structured geocoding is what we're calling the ability to provide the geocoding engine with the address broken up into its constituent parts. We've just released this functionality into production on our hosted instance of Pelias, known as Mapzen Search. *If you've been following our **release notes** (<http://pelias.io/release-notes.html>), we've been brewing this in beta for a few weeks.* At last, structured geocoding is now available at <http://search.mapzen.com/v1/search/structured> .

Before you run off and start **doing less** (<https://www.youtube.com/watch?v=PKIpCPS-oZc>), we invite to do one last thing and read on about where, when, and how to use this awesome new feature.

One Parameter Doesn't Fit All



Skylab Tool Kit #2, via **Cooper Hewitt** (<http://cprhw.tt/o/48Jdc/>)

Until now, Mapzen has only supported geocoding and searching using a single text input that contained all the search and location data. Sometimes this isn't the best way to geocode since your application's needs may not have information in this format. Consider a **CSV** (https://en.wikipedia.org/wiki/Comma-separated_values) file full of addresses to be geocoded:

address	city	state	country
1600 Pennsylvania Ave	Washington	DC	US
10 Downing Street	London		GB
55 Rue du Faubourg Saint-Honoré	Paris		FR

address	city	state	country
Bulevardul Geniului 1	Bucharest		Romania

Or, in another use case, say for some reason you recently decided to **move to Canada** (<http://www.wikihow.com/Move-to-Canada>) and your GPS device needs to geocode your new home address. The ambiguity presented by a single text input isn't ideal in this situation, so you'll most likely be presented with a multi-field prompt in which to enter your new address:

address	city	state	country
9 Queen Elizabeth Way	Fort Erie	ON	CA

Without separate fields the application would have to concatenate the address parts together into a single input to Pelias. While this task may seem fairly pedestrian, this is problematic for several reasons.

First, ambiguity can be introduced with concatenation. For example, `10 Park Place North Charleston South Carolina` can be legitimately interpreted by an address analyzer as a city name containing a directional (**North Charleston** (<https://whosonfirst.mapzen.com/spelunker/id/101720751/>) actually is a city in South Carolina):

```
{
  "address": "10 Park Place",
  "locality": "North Charleston",
  "region": "South Carolina"
}
```

or as a street with a post-directional:

```
{
  "address": "10 Park Place North",
  "locality": "Charleston",
  "region": "South Carolina"
}
```

Second, it's not always clear *how* to concatenate the fields of an address. The United States places the zip code after the state (e.g. 801 Leroy Place, Socorro, NM 87801) whereas Germany formats addresses with the postal code between the street address and city (e.g. Otto-Dürr-Straße 1, 70435 Stuttgart, Germany).

Third, the **dizzying variety of address formats** (<https://github.com/OpenCageData/address-formatting/blob/master/conf/countries/worldwide.yaml>) and **edge cases** (<https://www.mjt.me.uk/posts/falsehoods-programmers-believe-about-addresses/>) mean that address parsing is tricky business. We use **libpostal** (<https://github.com/openvenues/libpostal>) for text parameter parsing at our /v1/search endpoint and, while it's a fantastic address parser, a geocoder should not introduce ambiguity where application data contains little to none.

In any case, forcing the user to concatenate multiple fields into one for single input geocoding puts an undue burden upon the application developer.

Parameters

As the name hopefully implies, *structured* geocoding means the requesting application has geographic data already split up into its constituent parts. Structured geocoding has been deployed to the /v1/search/structured endpoint and accepts one or more of the following parameters:

- address
- neighbourhood
- borough
- locality
- county
- region
- postcode
- country

Using these parameters, you can construct requests that geocode full addresses or just a city and country, for example.

Along with the new parameters, structured geocoding supports all the other **search parameters** (<https://mapzen.com/documentation/search/search/#available-search-parameters>) that you've grown to love, like `boundary.country` , `sources` , `layers` , and

size .

You can find examples of how to construct requests for each of these components in the **structured geocoding documentation** (<https://mapzen.com/documentation/search/structured-geocoding/>).

address

The `address` parameter can contain a full address including house number or just a street name. Pelias stores addresses as separate number and street fields (libpostal is utilized to parse the number and street values from the address field).

neighbourhood

Neighbourhoods

(<https://whosonfirst.mapzen.com/spelunker/placetypes/neighbourhood/>) are vernacular geographic entities that may not necessarily be official administrative divisions but are important nonetheless.

borough

Boroughs (<https://whosonfirst.mapzen.com/spelunker/placetypes/borough/>) are a bit of an oddity in the realm of spatial data. For the most part they fit in between neighbourhoods and localities but are mostly identifiable to the general public in the context of New York City even though other cities such as **Mexico City** (<https://whosonfirst.mapzen.com/spelunker/id/857683023/descendants/?exclude=nullisland&placetype=borough>) have them, too. In fact, they're commonly thought of as cities themselves rather than as subsidiaries of **New York City** (https://whosonfirst.mapzen.com/spelunker/placetypes/borough/?®ion_id=85688543).

We don't expect our users to understand or appreciate the hierarchical distinction in our data between boroughs and localities, so if a structured geocode request passed `/v1/search/structured?locality=Manhattan®ion=NY` , Pelias will search boroughs along with localities.

locality

Localities (<https://whosonfirst.mapzen.com/spelunker/placetypes/locality/>) are equivalent to what are commonly referred to as cities, but can range anywhere in size from the **smallest hamlet** (<https://whosonfirst.mapzen.com/spelunker/id/85977019/>) to the **most populous metropolis** (<https://whosonfirst.mapzen.com/spelunker/id/102031307/>) on the planet.

county

Counties (<https://whosonfirst.mapzen.com/spelunker/placetypes/county/>) are administrative divisions between localities and regions.

Counties are not as commonly-used in geocoding as localities but can be useful when attempting to disambiguate between localities. For instance, there are 3 cities named Red Lion in Pennsylvania but only 1 in each of 3 counties. Specifying a county disambiguates this list to a single result.

region

Regions (<https://whosonfirst.mapzen.com/spelunker/placetypes/region/>) are normally the first-level administrative divisions within countries, analogous to states and provinces in the **United States** (<https://whosonfirst.mapzen.com/spelunker/id/85633793/descendants/?exclude=nullisland&placetype=region>) and **Canada** (<https://whosonfirst.mapzen.com/spelunker/id/85633041/descendants/?exclude=nullisland&placetype=region>), respectively, though most other countries contain regions as well.

Regions in the United States have **common abbreviations** (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations), such as PA for **Pennsylvania** (<https://whosonfirst.mapzen.com/spelunker/id/85688481/>) and NM for **New Mexico** (<https://whosonfirst.mapzen.com/spelunker/id/85688493/>). The `region` parameter can be a full name or abbreviation, so specifying `/v1/search/structured?region=NM` is functionality equivalent to `/v1/search/structured?region=New Mexico`.

postalcode

Postal codes (<https://whosonfirst.mapzen.com/spelunker/placetypes/postalcode/>) are used to aid in sorting mail with the format dictated by an administrative division (almost always countries). Among other reasons, postal codes are unique within a country so they're useful in geocoding as a shorthand for a fairly granular geographical location.

Pelias doesn't currently import postal codes, though addresses from **OpenAddresses** (<https://openaddresses.io/>) and **OpenStreetMap** (<https://www.openstreetmap.org/>) are sometimes annotated with postal codes and used for scoring.

country

Countries (<https://whosonfirst.mapzen.com/spelunker/placetypes/country/>) are the highest-level administrative divisions supported by Pelias. In addition to full names, countries have common **2-** (https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2) and **3-letter** (https://en.wikipedia.org/wiki/ISO_3166-1_alpha-3) abbreviations which are also supported values for the `country` parameter.

The more astute geocoding blog reader that has spent entirely too much time perusing **Who's on First** (<https://whosonfirst.mapzen.com/>) would have noticed that **Bermuda** (<https://whosonfirst.mapzen.com/spelunker/id/85632731/>) is actually a **British Overseas Territory** (https://en.wikipedia.org/wiki/British_Overseas_Territories) (a `dependency`, in Who's on First nomenclature) and not a country, similar to the relationship **Puerto Rico** (<https://whosonfirst.mapzen.com/spelunker/id/85633729/>) and **New Caledonia** (<https://whosonfirst.mapzen.com/spelunker/id/85632473/>) have to the **United States** (<https://whosonfirst.mapzen.com/spelunker/id/85633793/descendants/?exclude=nullisland&placetype=dependency#4/2.92/-170.78>) and **France** (<https://whosonfirst.mapzen.com/spelunker/id/85633147/>), respectively. To reduce the number of parameters and potential confusion about data organization, **dependencies** (<https://whosonfirst.mapzen.com/spelunker/placetypes/dependency/>) are searched for using the `country` parameter value.

Caveats

Any combination of the above parameters can be sent as structured geocoding requests **with the exception of postalcode-only** as Pelias does not currently import postal codes as separate records, only as augmenting address data. For example, a request consisting only of `/v1/search/structured?postalcode=87801` is not valid at this time and an error will be returned to the caller.

Fallback Behaviors

Structured geocoding, much like the single-input `/v1/search` endpoint, falls back to less granular geocodes if the exact input as specified returns no results. This topic has already been covered in **The Next Chapter of Search** (<https://mapzen.com/blog/the-next-chapter-of-search/>) so it won't be covered here except for a quick recap.

A key concept of geocoding is to not return things other than what the user asked for. If the geocode request is for `14 Horseshoe Pond Lane, Concord, New Hampshire` and that address neither is a point in the data nor can it be interpolated, a geocoder shouldn't return something that's close like `16 Horseshoe Pond Lane, Concord, New Hampshire`. The geocoder should fall back to the most granular level available. If a street result for `Horseshoe Pond Lane, Concord, New Hampshire` is in the data, return that. Otherwise return `Concord, New Hampshire` or even `New Hampshire` as a last resort.

Who's On First Layer Mappings

This section is for people who are well-versed in the nuances of Who's on First place types in or have spent a bit of time looking at data in it.

As stated previously, we don't expect our users to understand the **complexities of Who's on First layer mappings** (<https://whosonfirst.mapzen.com/placetypes/>). While there are very good reasons why our gazetteer supports both `locality` and `localadmin`, it would be pretty cumbersome to include both as parameters, so we have added some convenience mappings to make structured geocoding easier:

structured geocoding parameter	Who's on First placetype(s)
<code>neighbourhood</code>	neighbourhood (https://whosonfirst.mapzen.com/spelunker/placetypes/neighbourhood/)
<code>borough</code>	borough (https://whosonfirst.mapzen.com/spelunker/placetypes/borough/)
<code>locality</code>	locality (https://whosonfirst.mapzen.com/spelunker/placetypes/locality/), localadmin (https://whosonfirst.mapzen.com/spelunker/placetypes/localadmin/) (and borough (https://whosonfirst.mapzen.com/spelunker/placetypes/borough/) if <code>borough</code> parameter is not supplied)
<code>county</code>	county (https://whosonfirst.mapzen.com/spelunker/placetypes/county/), macrocounty (https://whosonfirst.mapzen.com/spelunker/placetypes/macrocounty/)
<code>region</code>	region (https://whosonfirst.mapzen.com/spelunker/placetypes/region/), macroregion (https://whosonfirst.mapzen.com/spelunker/placetypes/macroregion/)

structured geocoding parameter	Who's on First placetype(s)
country	dependency (https://whosonfirst.mapzen.com/spelunker/placetypes/dependency/), country (https://whosonfirst.mapzen.com/spelunker/placetypes/country/)

For example, **Peach Bottom, Pennsylvania**

(<https://whosonfirst.mapzen.com/spelunker/id/404487863/>) is only a `localadmin` place type and not a `locality` in Who's on First, but we don't expect the user to know the distinction, so if a structured geocoding request specifies `locality=Peach Bottom®ion=Pennsylvania`, then Pelias will lookup `Peach Bottom` in both the `locality` and `localadmin` layers.

Get in touch!

Check out the structured geocoding **documentation**

(<https://mapzen.com/documentation/search/structured-geocoding>), and if you have any questions, concerns, enhancement requests, or bug reports, please don't hesitate to **file an issue** (<https://github.com/pelias/pelias/issues>)! Get more get more geocoding with less work!

Note: This post was updated on May 31, 2017 to update the API request links. In addition, since this post was written, postalcode searches are supported.

· 07 December 2016 ·

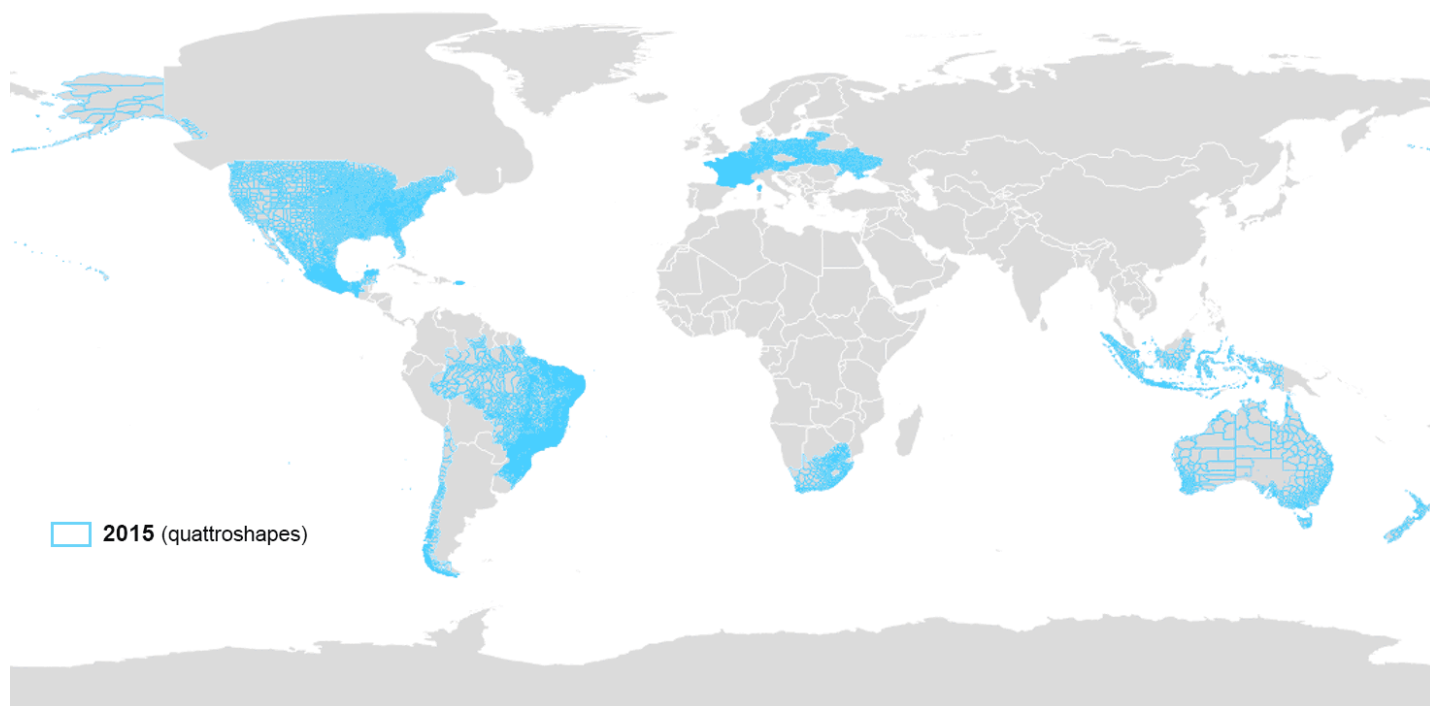


Stephen Hess

Stephen works on geocoding exclusively as a means to fund his passion for designing and building wooden frames for old maps.

Improving county coverage in Who's On First

`whosonfirst (/tag/whosonfirst)` `data (/tag/data)`



We've doubled the number of `county` features in **Who's On First** (<https://whosonfirst.mapzen.com>), a gazetteer of places all based on open data.

When we first started the gazetteer, 18,214 `counties` were imported from **Quattroshapes** (<http://quattroshapes.com>), a compendium of open data from national mapping agencies.

We've grown that number to **41,275**

(<https://whosonfirst.mapzen.com/spelunker/placetypes/county/#4/-1.97/-47.02>) `counties` in 2016 by adding open data from additional CC-BY sources, and by creating new shapes that Mapzen is releasing as CC0.

We're on track to having the first high-quality, open-licensed dataset of `county` polygons that combines global coverage with a permissive license to enable developers to build commercial applications.

Why are county polygons important to Who's On First?

The goal of Who's On First has always been to build a gazetteer of places, with each place given a stable identifier and some number of descriptive properties about that location.

The goal of importing new `county` features is to upgrade shapes for existing places and to achieve global coverage by adding additional places.

Sourcing additional county polygons

In many cases, new `county` data (sometimes called administrative level 2, or adm-2 for short) was imported from national mapping agencies under a CC-BY compatible license with little to no modification into Who's On First. (Our collective advocacy for open data is paying off, thank you!)

For example, the Taiwanese government maintains openly licensed geodata for nearly all **categorized placetypes** (<https://github.com/whosonfirst/whosonfirst-placetypes/blob/master/README.md>) in Who's On First. Once data for Taiwan was retrieved, property names in the source were mapped to standardized Who's On First property names and imported using the techniques described below.

In other cases, when openly licensed `county` data could not be found for a country, Mapzen drew polygons to fill in the gaps. To draw these geometries we compared three or more reference datasets to create our own shapes. In keeping with the lineage of alphashapes, betashapes, quattroshapes, zetashapes, and yerbashapes, we call these new geometries **mesoshapes**, and you'll see `meso` in the feature's `src:geom` property.

Mesoshapes

When new `county` polygons were required, Mapzen drew them using a semi-automated GIS process:

- seed a set of points across the entire country
- discard points within a buffer of the reference's boundary line geometries
- for each point, compare the reference sources to determine if there is consensus about which county the point falls in
- create **Thiessen polygons** (<https://pro.arcgis.com/en/pro-app/tool-reference/analysis/create-thiessen-polygons.htm>) around each consensus point
- dissolve Thiessen polygons based on the consensus county attribute
- generalize the resulting polygon shapes

Some manual cleanup was performed, but not a lot. In our review, areas with good consensus between references on the essential facts will result in good polygons – and this is true for the majority of our results. This process allowed us to import new county polygons for countries that do not currently offer open data.

How were new county polygons imported?

Once new county features were inventoried, an import strategy was needed to handle all geometries, associated properties, and potential conflicts. As a first step, each new county feature was compared to all existing county features for the appropriate country. This was done by:

- Comparing names of the county feature with name properties of existing Who's On First features.
- Ensuring the county feature's parent (likely a region) matched the existing Who's On First feature's parent. Many countries share names among several county features; comparing names *and* locations are equally important.

If this resulted in no matches, the county feature was imported as a new feature to Who's On First. If a match was found, the following considerations were needed:

- Adding properties from the new county feature properties to existing Who's On First record
- Retaining existing Who's On First record properties
- Handling all variations of names and name variants during the merge
- Storing the existing feature's outdated geometry as an alt-geometry file
- Attaching the new geometry to the existing feature while updating any geometry-related property

Additionally, since we know all new county features are contiguous and inclusive of all counties in a given country, we needed to deprecate any “leftover” features in Who's On First that did not match any incoming county features.

Comparing records pre-import and post-import

Let's compare the county features in Afghanistan, pre-import and post-import. Before importing new county geometries, Who's On First maintained records for 25 counties in Afghanistan, all of which were represented by point geometries. There were 400 new county polygon features created for Afghanistan; each of these features were compared to the 25 existing county records in Afghanistan.

Below, the county records prior to importing counties are shown as yellow points. The new county polygons are displayed as polygons (blue outline). Zoom in to see all new county names, or **view full screen** (https://tangrams.github.io/tangram-frame/?minz=5&maxz=10&url=https://s3.amazonaws.com/whosonfirst.mapzen.com/misc/_blogs/mesoshapes_blogpost.yaml&maxbounds=29.377217,60.475769,38.490733,74.889862#5.5/34.179/65.126).



This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.

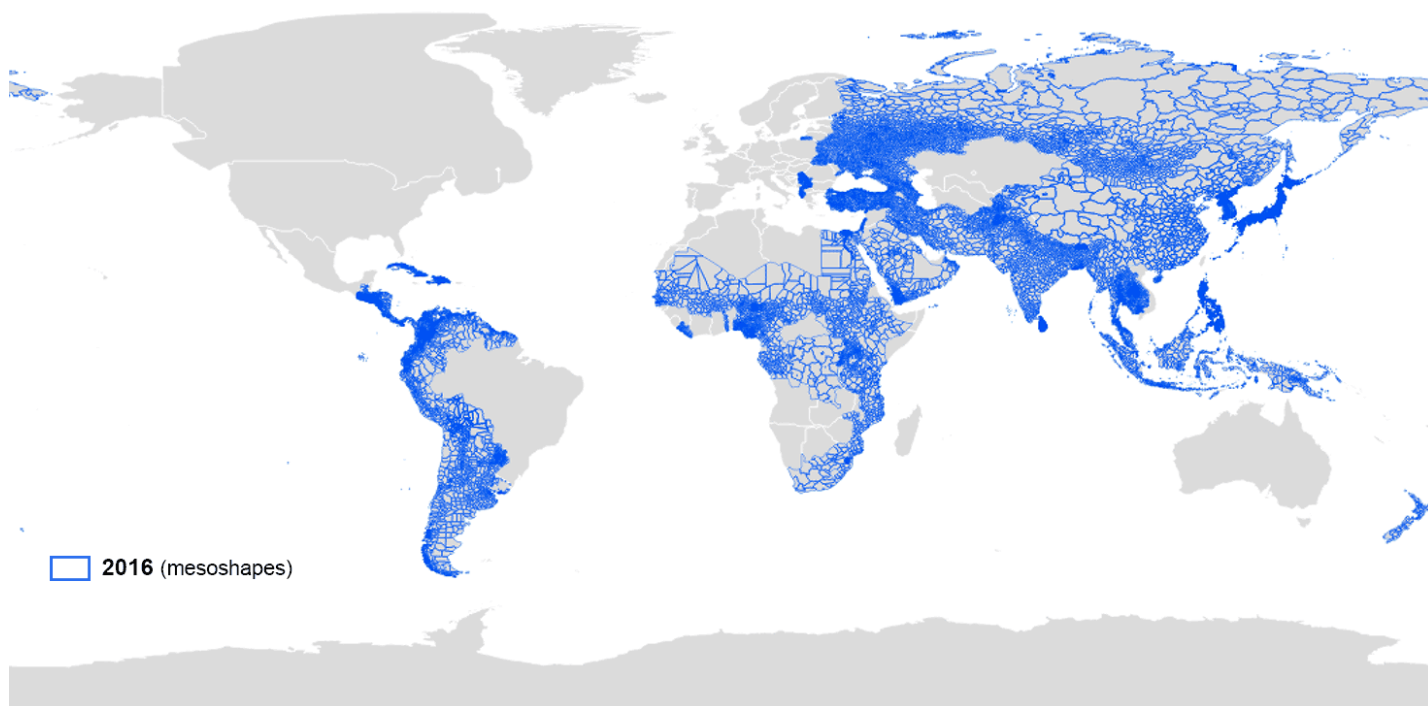
[Learn more.](#)

Prior to the import of new county features, Who's On First had a total of **8,172** mostly point geometry based county features for the countries included in the first round of countries (added earlier in 2016 and not included in the animated map).

After the import, Who's On First contains **21,469** county features for those same countries - an increase of **13,297** county features. **Forty** of these countries had virtually no county coverage in Who's On First, but now have high-quality county records available for use or

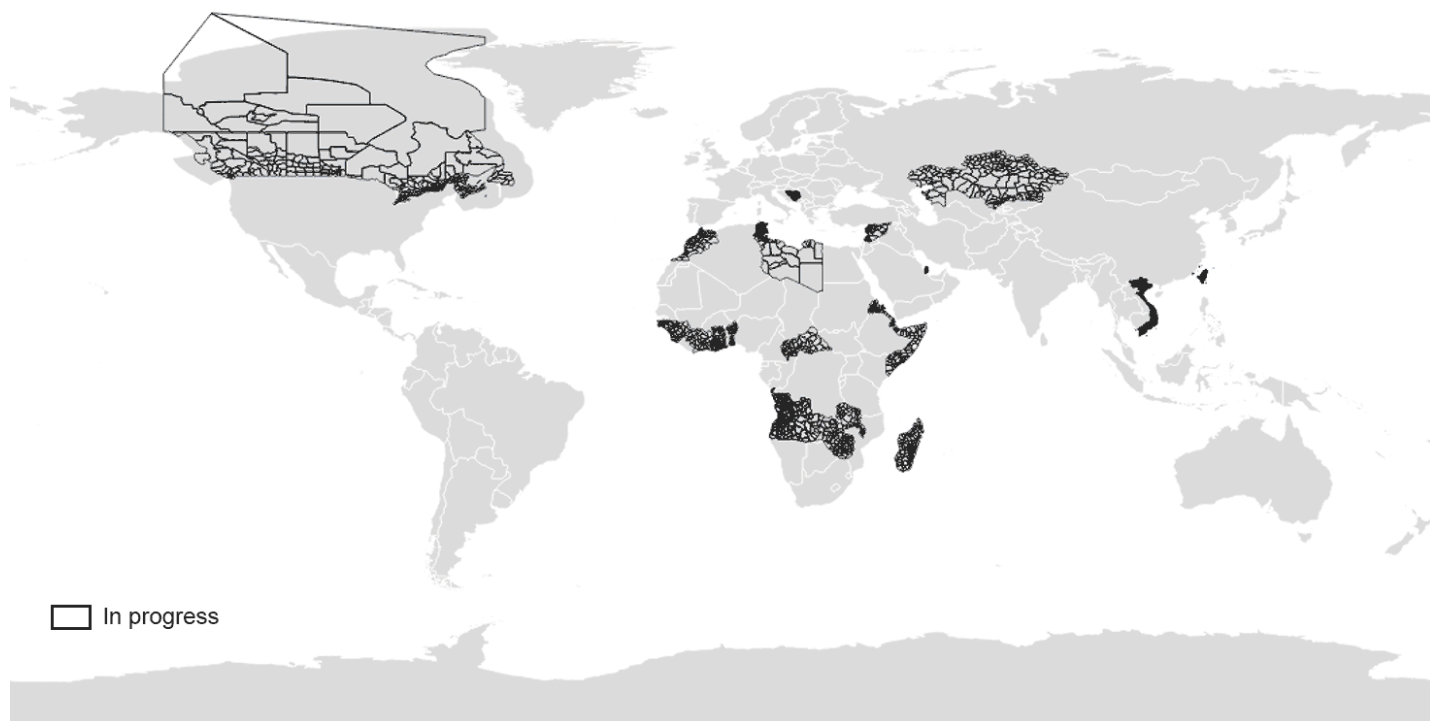
download.

In rare cases (including Chile, Indonesia, New Zealand, South Africa, and the Ukraine), the earlier 2015 data was found to be poor-quality and was upgraded with high-quality data.



Next steps...

While Who's On First has already imported a substantial number of new county features, we plan on importing additional county features through early 2017. So far we've processed an additional 3,353 features over today's announcement (show in black outlines on the map below). That leaves Canada and a handful of countries in Africa and Asia (ignoring small island countries for now). In our research we've found many European countries skip the county administrative level and go straight to `localadmin`, but please send us data sources, tips, and corrections!



We found a few other data gems along the way, so expect related improvements to other placetypes including `locality` and `region`.

Most new `county` feature includes a unique Statoids **HASC code** (<http://www.statoids.com/ihasc.html>) in their properties. Besides joining the HASC code with demographic tables to create data visualizations, you could use the code to dissolve `county` geometries into new `region` polygons that are often better quality than the existing `region` polygon in Who's On First. The following HASC codes, as an example, would be used to create the `AF.AR` region shape (with a little help from QGIS).

```
AF.AR.EC
AF.AR.PL
AF.AR.FG
AF.AR.RT
```

And there you have it - new county polygons in Who's On First! Thanks for reading and stay tuned for the next rounds of new administrative polygons!

Please ***let us know what you think*** (<mailto:hello@mapzen.com>)!



Stephen Epps

Stephen is Mapzen's gardener and fixer-upper of geographic data.



Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.



Martin Gamache

Martin Gamache is a cartographer and geographer with Art of the Mappable.

© 2017 Mapzen

Interactive Mapping with Tangram

[mapzen-js \(/tag/mapzen-js\)](#) [demo \(/tag/demo\)](#) [tutorial \(/tag/tutorial\)](#)

[tangram \(/tag/tangram\)](#) [make-your-own \(/tag/make-your-own\)](#)

Hi there.

We're back with another installment of **Make Your Own** (<https://mapzen.com/tag/make-your-own/>) [[Interactions](#)].

Previously, we've looked at how our data can be filtered, styled, and displayed in a Tangram map. But what if you want to create a full **interactive mapping experience** for your users? Why, Tangram does that, too!

Today we're going to be looking at a few different ways we can interact with our data: **we'll make a simple layer toggle, add a popup on click, and add a highlighted boundary to a selected feature**. Yes, it's a lot, but just think of all the cool interactions you can add to your next Tangram map...

You'll really be bringing your map sandwich to life!

Make your map sandwich come alive (<https://vimeo.com/195536057>)!

We'll be building upon our work from **last time**

(<https://bl.ocks.org/rfriberg/5eebfa479ede198c9ba5a8545b838620>), so if you haven't yet read the first few posts, I recommend starting there:

- **One Minute Map** (<https://mapzen.com/blog/one-minute-map/>)
- **Map Sandwich** (<https://mapzen.com/blog/map-sandwich/>)
- **Filters & Functions** (<https://mapzen.com/blog/filters-and-functions/>)
- **Put A Label On It** (<https://mapzen.com/blog/tangram-labels/>)

In the last few posts, we focused primarily on the Tangram scene file (typically called `scene.yaml`). In today's post, we'll be focusing more heavily on the JavaScript side of things and will largely be working within our `index.html` file.

Ready to jump in? ... *That's the spirit!*

We'll begin with the following two files.

index.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>San Juan Island Geology – Interactivity</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="https://mapzen.com/js/mapzen.css">
    <script src="https://mapzen.com/js/mapzen.min.js"></script>
    <style>
      #map {
        height: 100%;
        width: 100%;
        position: absolute;
      }
      html,body{margin: 0; padding: 0}
    </style>
  </head>
  <body>
    <div id="map"></div>
    <script>
      // Mapzen API key (replace key with your own)
      // To generate your own key, go to https://mapzen.com/developers/
      L.Mapzen.apiKey = 'mapzen-JA21Wes';

      var san_juan_island = [48.5326, -123.0879];

      var map = L.Mapzen.map('map', {
        center: san_juan_island,
        zoom: 13,
        scene: 'scene.yaml',
      });

      // Add attribution for NPS geology data
      map.attributionControl.addAttribution('Geology data &copy; <a href="https://www.nps.gov/geology">NPS Geology</a>');

      // Mapzen Search box (replace key with your own)
      // To generate your own key, go to https://mapzen.com/developers/
      var geocoder = L.Mapzen.geocoder('mapzen-JA21Wes');
      geocoder.addTo(map);

      // ADD LAYER TOGGLE

      // ADD SELECTION EVENTS

      // ADD HIGHLIGHT

      // ADD TANGRAMLOADED LISTENER

    </script>
```

```
</body>  
</html>
```

*UPDATE March 1, 2017: A Mapzen developer API key is now required for mapzen.js. We've updated the Make Your Own series to include a demo key. Generate your own free API key at **<https://mapzen.com/developers/>** (**<https://mapzen.com/developers/>**).*

and **scene.yaml**:

```

import: https://mapzen.com/carto/walkabout-style/3/walkabout-style.zip

styles:
  _alpha_polygons:
    base: polygons
    blend: multiply
  _dashed_lines:
    base: lines
    dash: [3, 1]
    dash_background_color: rgb(149, 188, 141)

sources:
  _nps_boundary:
    type: GeoJSON
    url: https://gist.githubusercontent.com/rfriberg/684645c22f495b4a46f29fb312b6d268,

  _nps_geology:
    type: GeoJSON
    url: https://gist.githubusercontent.com/rfriberg/3c09fe3afd642224da7cd70aff1c1e70,
    generate_label_centroids: true

layers:
  _national_park:
    data: { source: _nps_boundary }
    draw:
      _dashed_lines:
        width: [[8, 0.5px], [18, 5px]]
        color: '#518946'
        order: global.sdk_order_over_everything_but_text_1

  _geology:
    data: { source: _nps_geology }
    filter:
      all:
        - { $zoom: { min: 10 } }
        - not: { GLG_SYM: water }
    draw:
      _alpha_polygons:
        order: global.sdk_order_over_everything_but_text_0
        color: |
          function() {
            // Note: this is a block of JavaScript so we can use JS comment s
            var category = feature.GLG_SYM;
            var color = category == 'Qa'      ? '#FFF79A' :
                                category == 'Qb'      ? '#FFF46E' :
                                category == 'Qd'      ? '#fff377' :
                                category == 'Qf'      ? '#dddddd' :
                                category == 'Qp'      ? '#EAC88D' :

```



```

        category == 'Qgdm'      ? '#FCBB62' :
        category == 'Qgdm(es)' ? '#FEE9BB' :
        category == 'Qgdm(e)'  ? '#E8A121' :
        category == 'Qgom(e)'  ? '#EAB564' :
        category == 'Qgom'     ? '#FECE7A' :
        category == 'Qgd'      ? '#FEDDA3' :
        category == 'Qgt'      ? '#FCBB62' :
        category == 'KJmm(c)'  ? '#86C879' :
        category == 'KJm(ll)' ? '#9FD08A' :
        category == 'JTRmc(o)' ? '#27BB9D' :
        category == 'TRn'     ? '#ED028C' :
        category == 'TRPMv'   ? '#F172AC' :
        category == 'TRPv'    ? '#F499C2' :
        category == 'PDmt'    ? '#40C7F4' :
        category == 'pPsh'    ? '#9BA5BE' :
        category == 'pDi'     ? '#848FC7' :
        category == 'pDit(t)' ? '#B28ABF' :
        '#000';

    return color;
}

_geology_labels:
  filter: { label_placement: true, $zoom: { min: 13 } }
  draw:
    text:
      text_source: GLG_SYM
      font:
        fill: rgba(130, 84, 41, 0.9)
        size: [[13, 10px], [20, 24px]]
        weight: bold
        stroke:
          color: rgba(242, 218, 193, 0.3)
          width: 3

# Override Walkabout's layers
landuse:
  visible: false
roads:
  minor_road:
    visible: false

```

I made a couple of changes to our `index.html` file since you've seen it last: I removed the custom positioning of the zoom control and added an attribution for our **geology data** (<https://www.nps.gov>). But for the most part, it should look much as you remember it from our **last post** (<https://mapzen.com/blog/tangram-labels/>).

Layer Switcher

Let's start by creating a simple layer switcher for our geology layer. It will be nothing more than a checkbox that toggles the visibility of our geology layer, but that seems like a good place to start.

We're going to extend **Leaflet** (<http://leafletjs.com/>)'s **Control class** (<http://leafletjs.com/reference.html#control>) to create our switcher. This will allow us to use Leaflet's positioning option as well as a few built-in methods.

In the `<script>` tag of your `index.html` file, add the following under `// ADD LAYER TOGGLE` :

```
/// ADD LAYER TOGGLE
var LayerToggleControl = L.Control.extend({
  options: {
    position: 'topright'
  },

  onAdd: function () {
    var container = L.DomUtil.create('div', 'layer-control');
    container.innerHTML = '<label><input id="layer_toggle" type="checkbox" checked>';

    return container;
  },

});
var toggleControl = new LayerToggleControl();
map.addControl(toggleControl);
```

This will add a checkbox and label to the top right corner of your map.

*Note: the `onAdd` function we're using is a **built-in method** (<http://leafletjs.com/reference.html#control-onadd>) that Leaflet calls once the control is added to the map.*

To help our checkbox stand out against the map, add this css to your `<style>` section at the top of your page:

```
.layer-control {
  background: rgba(255, 255, 255, 0.85);
  padding: 0 10px;
  border-radius: 5px;
}
```

Now let's make the control actually *do* something. In the `onAdd` method, we'll add a listener to listen for clicks. Then add a click handler (called `toggleOnClick`) to process those clicks:

```
// ADD LAYER TOGGLE
var LayerToggleControl = L.Control.extend({
  options: {
    position: 'topright'
  },

  onAdd: function () {
    var container = L.DomUtil.create('div', 'layer-control');
    container.innerHTML = '<label><input id="layer_toggle" type="checkbox" checked>';

    L.DomEvent.on(container, 'click', this.toggleOnClick);
    return container;
  },

  toggleOnClick: function (e) {
    var showLayer = document.getElementById('layer_toggle').checked;
    alert('show layer? ' + showLayer);
  }
});
```

In this example, all we're doing is checking whether our checkbox is checked. We'll use this value to determine whether we should show (`true`) or hide (`false`) our layer.

Ok, now we need a way to tell Tangram whether or not we want to display this particular layer. We'll do that by setting the **visibility parameter**

(<https://mapzen.com/documentation/tangram/draw/#visible>) for that layer. You may remember this parameter from a previous post, when we temporarily hid our national park boundary layer in our scene file. There is also a way to update this parameter using JavaScript.

To do so, we'll need access to Tangram's **scene object**

(<https://mapzen.com/documentation/tangram/Javascript-API/#scene>).

Currently, mapzen.js fires a **“tangramloaded” event**

(<https://mapzen.com/documentation/mapzen-js/api-reference/#events>) once Tangram is loaded and available to our page. We'll add a listener to listen for this event, and grab the scene object once it's available. To make sure the scene object is available outside of this listener, we'll assign it to a global variable called `scene`. (*I'm sure that's not confusing at all.*)

At the bottom of our `<script>` section, where it says `// ADD TANGRAMLOADED LISTENER`, let's add our global `scene` variable and the listener:

```
// ADD TANGRAMLOADED LISTENER
var scene;
map.on('tangramloaded', function(e) {
  var tangramLayer = e.tangramLayer;
  scene = tangramLayer.scene;
});
```

Great. Now back in our custom Leaflet control, we can use this `scene` object in our `toggleOnClick()` method. Tangram (and its scene object) loads pretty fast, but we'll still add a simple check to make sure it's loaded and handle any quick-clicker friends who may be viewing our map:

```
toggleOnClick: function (e) {
  var showLayer = document.getElementById('layer_toggle').checked;

  if (scene) {
    // do something
  }
}
```

Now that we have the `scene` object available, we can do quite a bit to our Tangram map. In fact, here's the **scene object documentation**

(<https://mapzen.com/documentation/tangram/Javascript-API/#scene>) once more, in case you want to read up on all of the methods and properties you now have access to.

The `scene` property we'll use the most is the `config` object. The **config object** (<https://mapzen.com/documentation/tangram/Javascript-API/#config>) is the JavaScript object version of our *entire scene file*. So, anything we could update in our scene file can also be accessed via the `scene.config` JavaScript object.

For instance, if we want to change the visibility of our geology layer, we could make the change in the scene file:

```
layers:
  # San Juan geology layer (polygons and labels)
  _geology:
    visible: false
    filter: ...
    draw: ...
```

Or we can do so dynamically using JavaScript:

```
scene.config.layers._geology.visible = false;
```

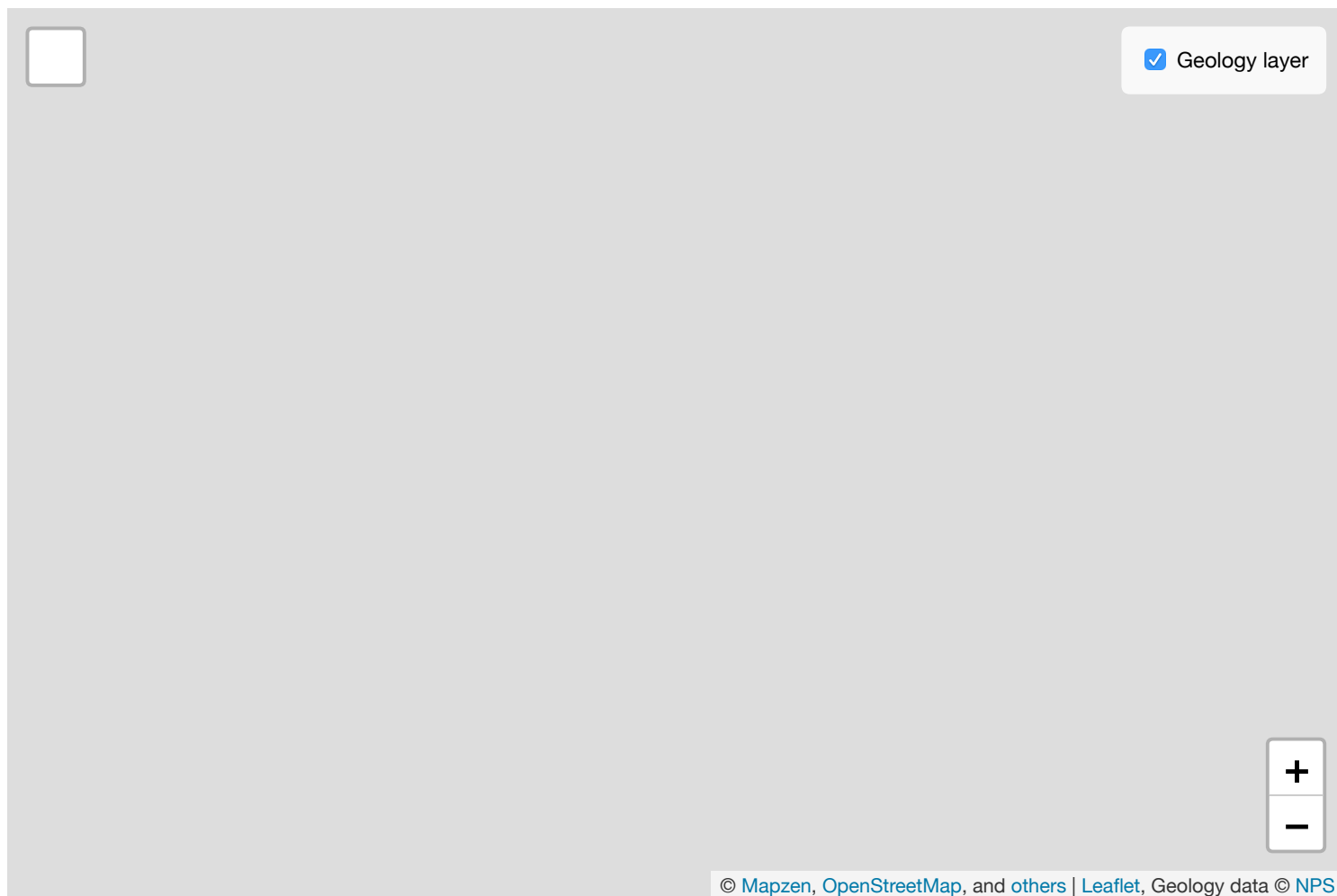
Note: the same method works on sublayers, as well:

```
scene.config.layers._geology.draw._geology_labels.visible = false;
```

After making changes to `scene.config`, we then call `scene.updateConfig()` to update the scene with all of our changes. So, putting it all together, we now have a `toggleOnClick` method that will hide and show our geology layer based on a checkbox click:

```
toggleOnClick: function (e) {
  var showLayer = document.getElementById('layer_toggle').checked;

  if (scene) {
    scene.config.layers._geology.visible = showLayer;
    scene.updateConfig();
  }
}
```



Honestly. I'd call that a successful day. You could knock off right here and rest happily with your newfound knowledge of how to interact with a Tangram map. Go ahead. Tell your boss I gave you the rest of the day off.

Ah, right. Popups *do* sound fun, don't they? Being able to dig out data from our layer and display it in a popup (or a tooltip or any other html element)... who really wants to miss that? **Not me!**

Popups

We'll start by adding a basic **Leaflet popup** (<http://leafletjs.com/reference.html#popup>) to the top of our script section:

```
var popup = L.popup();
```

This won't show up on your map, as it needs a location (at minimum), which we'll set later. For now, we just want a generic popup available.

Next, we'll set up a click handler to capture any map clicks. Tangram provides two **selection event handlers** (<https://mapzen.com/documentation/tangram/Javascript-API/#hover-and-click>): *hover* and *click*. In our case, the easiest way to add selection events is within our “tangramloaded” listener, when we have access to the `tangramLayer` object. At that point, we can use Tangram's `setSelectionEvents()` method.

Selection Event: Click

Let's go ahead and add a `click` selection event to our “tangramloaded” listener:

```
// ADD TANGRAMLOADED LISTENER
var scene;
map.on('tangramloaded', function(e) {
  var tangramLayer = e.tangramLayer;
  scene = tangramLayer.scene;

  tangramLayer.setSelectionEvents({
    click: function(selection) {
      alert('Clicked!');
    }
  });
});
```

If you're following along, your map will now display a “Clicked!” alert any time you click on the map. I know it's not much. *Baby steps, Bob!*

Our callback function is merely an alert at the moment, but it will grow. So before it gets unwieldy, let's pull that function out of our listener and call it `onMapClick`.


```
// ADD SELECTION EVENTS
function onMapClick(selection) {
  alert('Clicked!');
}

// ADD TANGRAMLOADED LISTENER
var scene;
map.on('tangramloaded', function(e) {
  var tangramLayer = e.tangramLayer;
  scene = tangramLayer.scene;

  tangramLayer.setSelectionEvents({
    click: onMapClick
  });
});
```

Much better.

Now let's make our `onMapClick` handler actually handle our click:

```
// ADD SELECTION EVENTS
function onMapClick(selection) {
  if (selection.feature) {
    var latlng = selection.leaflet_event.latlng;
    var label = selection.feature.properties.GLG_SYM;
    showPopup(latlng, label);
  }
}

function showPopup(latlng, label) {
  popup
    .setLatLng(latlng)
    .setContent('<p>' + label + '</p>')
    .openOn(map);
}
```

What did I tell you? It grew fast, right? Let me walk you through what we just did.

Our select event callback function (which we've called `onMapClick`) accepts an object from Tangram that was "selected" when we clicked on it. The selected object (`selection`) looks like this:

```
{ feature, changed, pixel, leaflet_event }
```

From the `leaflet_event` property, we can get the latitude/longitude of our click. And from the `feature` property, we can get the list of GeoJSON properties for that feature. If you recall from our **Filters and Functions post** (<https://mapzen.com/blog/filters-and-functions/>), each of our geology polygons comes with a set of GeoJSON properties:

```
{
  "type": "Feature",
  "properties": {
    "FUID": 1,
    "GLG_SYM": "KJmm(c)",
    "SRC_SYM": "KJm(c)",
    "SORT_NO": 13.0,
    "NOTES": "NA",
    "GMAP_ID": 74832,
    "HELP_ID": "KJmm(c)",
    "SHAPE_Leng": 137.95199379300001,
    "SHAPE_Area": 954.47301117400002
  },
  "geometry": ...
}
```

Once again, we'll use the `GLG_SYM` property as the text to display in our popup.

With `latlng` and `label` in hand, we can now set the location of our popup, set the content of our popup, and display it on our map.

If you're *really* following along, you may notice that nothing is actually happening. Reload. Click on the map. Nothing happens. That's because there's one last thing we need to do. And that is to tell Tangram that our geology polygons should be ~~query-able~~... ~~queriable~~... queryable. (*Nailed it.*)

We can do that by setting the **interactive** (<https://mapzen.com/documentation/tangram/draw/#interactive>) parameter of our geology polygons. We don't need to update this dynamically, so let's add the new parameter directly to our `scene.yaml` file:

```

layers:
  # San Juan geology layer (polygons and labels)
  _geology:
    data: ...
    filter: ...
    draw:
      _alpha_polygons:
        interactive: true

```

This `interactive: true` parameter enables Tangram's selection capability for those objects. Now, when we reload our map and click on a geology polygon, we should be seeing a simple popup displaying the appropriate geology symbol. When you click on a feature that does not have `interactive: true` (e.g., the water), no object will be selected and so no popup will appear.

Selection Event: Hover

While we're on the subject of selection events, let's add a handler for hover events, too:

```

function onMapHover(selection) {
  // do something
}

// ADD TANGRAMLOADED LISTENER
var scene;
map.on('tangramloaded', function(e) {
  var tangramLayer = e.tangramLayer;
  scene = tangramLayer.scene;

  tangramLayer.setSelectionEvents({
    click: onMapClick,
    hover: onMapHover
  });
});

```

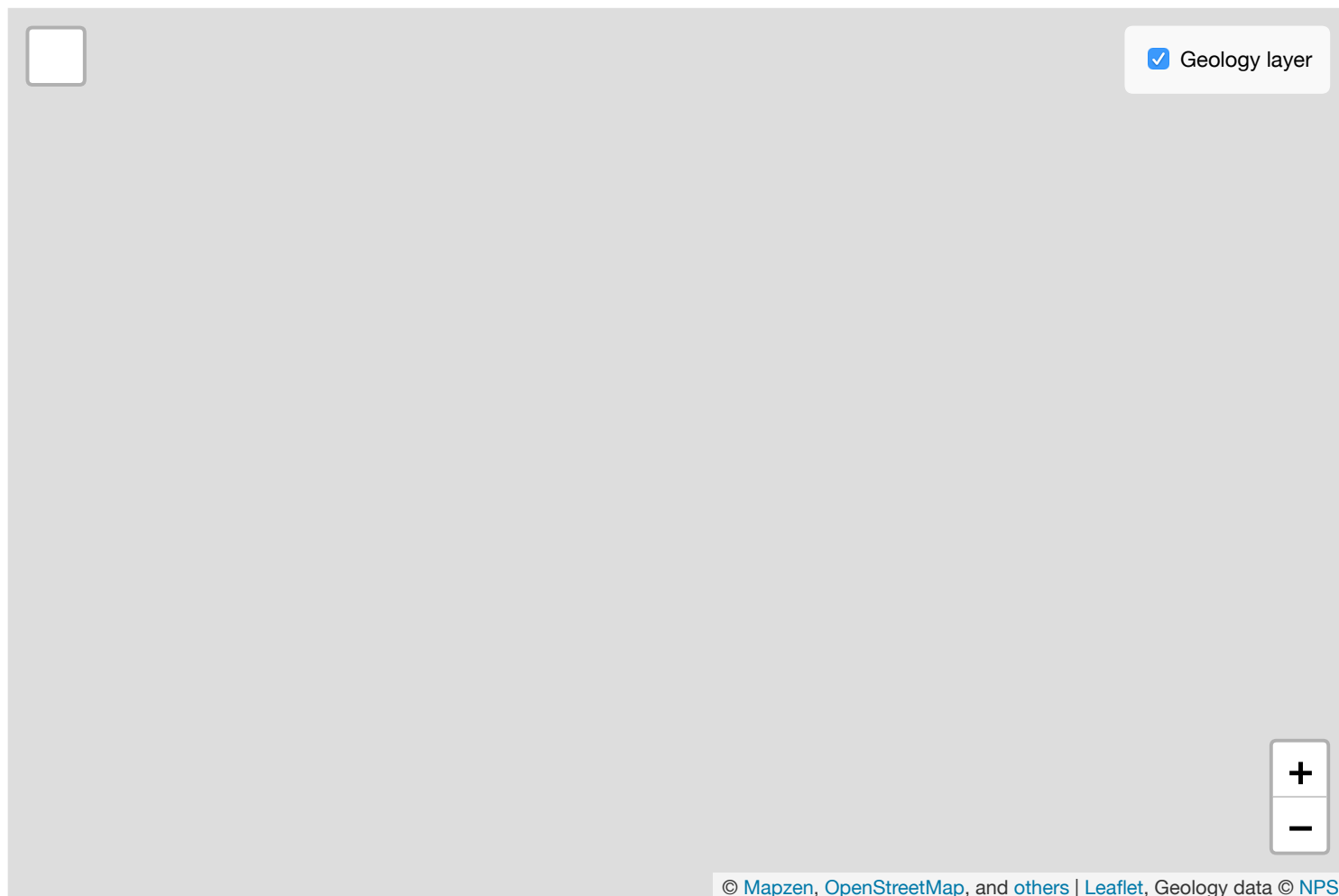
Now, every time our cursor *hovers* over the map, `onMapHover` will be called. It will be called **a lot**, so be careful in how you use this. (*With great power comes... well, you know the rest.*)

In this case, I just want to know when the cursor is hovering over a queryable object. We can do that with just a simple check for the `selection.feature` object:

```
function onMapHover(selection) {  
  document.getElementById('map').style.cursor = selection.feature ? 'pointer' : '';  
}
```

Now, when our cursor hovers over a queryable object, it will become a “pointer” and we’ll know it’s clickable.

Let’s see our *click* and *hover* events in action:



Feature Highlighting

Let’s look at one last thing before we wrap up. I’d like to add some highlighting to our features to help them stand out when they’ve been clicked. Currently, this requires a couple of steps to get working in Tangram.

We’ll start in our `scene.yaml` file.

Let's add a new **sublayer** (<https://mapzen.com/documentation/tangram/layers/#sublayer-name>) to our `_geology` layer to capture the style for our selected feature. We'll call it `_geology_highlight` :

```
_geology_highlight:
  draw:
    lines:
      width: 2px
      color: yellow
      order: global.sdk_order_over_everything_but_text_2
```

As it is now, this will display a yellow border around all of our geology polygons. To limit the style to a single polygon, we'll need to set up a filter. Let's take another look at those GeoJSON properties:

```
{
  "type": "Feature",
  "properties": {
    "FUID": 1,
    "GLG_SYM": "KJmm(c)",
    "SRC_SYM": "KJm(c)",
    "SORT_NO": 13.0,
    "NOTES": "NA",
    "GMAP_ID": 74832,
    "HELP_ID": "KJmm(c)",
    "SHAPE_Leng": 137.95199379300001,
    "SHAPE_Area": 954.47301117400002
  },
  "geometry": ...
}
```

The `FUID` property is a unique identifier. So, if we only want to highlight a single polygon, we should filter on that `FUID` property:

```
_geology_highlight:
  filter: { FUID: 108 }
  draw: ...
```

If you update your map now, you should see a single highlighted polygon in the center of the island.

But what if we want to highlight *all* polygons of the same rock type? We could do that by using the same `GLG_SYM` property we've used for our styling, labels, and popups:

```
_geology_highlight:  
  filter: { GLG_SYM: 'KJmm(c)' }  
  draw: ...
```

Now, when you reload, you'll see all of the green `KJmm(c)` polygons highlighted.

Great! Let's go with that. Some of those polygons are small and hard to see, so highlighting will be a good way to quickly see where each rock unit is exposed across the island and to facilitate the interpretation of the island's geologic structures.

I mean, if you're into that type of thing.

We'll move back to our JavaScript in just a second, but first, let's make it a little easier to update this filter. At the top of your `scene.yaml` file, add a **global variable** (<https://mapzen.com/documentation/tangram/global/>) called `_highlight` :

```
global:  
  _highlight: false
```

Note: I set the global variable to `false` so nothing will appear on first load, but really anything that's not a valid filter value will work here. If you want to test that it's working, go ahead and set it to `KJmm(c)` .

Then, update your filter to pull in this new global variable:

```
_geology_highlight:  
  filter: { GLG_SYM: global._highlight }  
  draw: ...
```

Now all we need to do from the JavaScript side of things is to update the global `_highlight` . This will be a little cleaner than passing in an entire filter object.

Ok. Back to `index.html` .

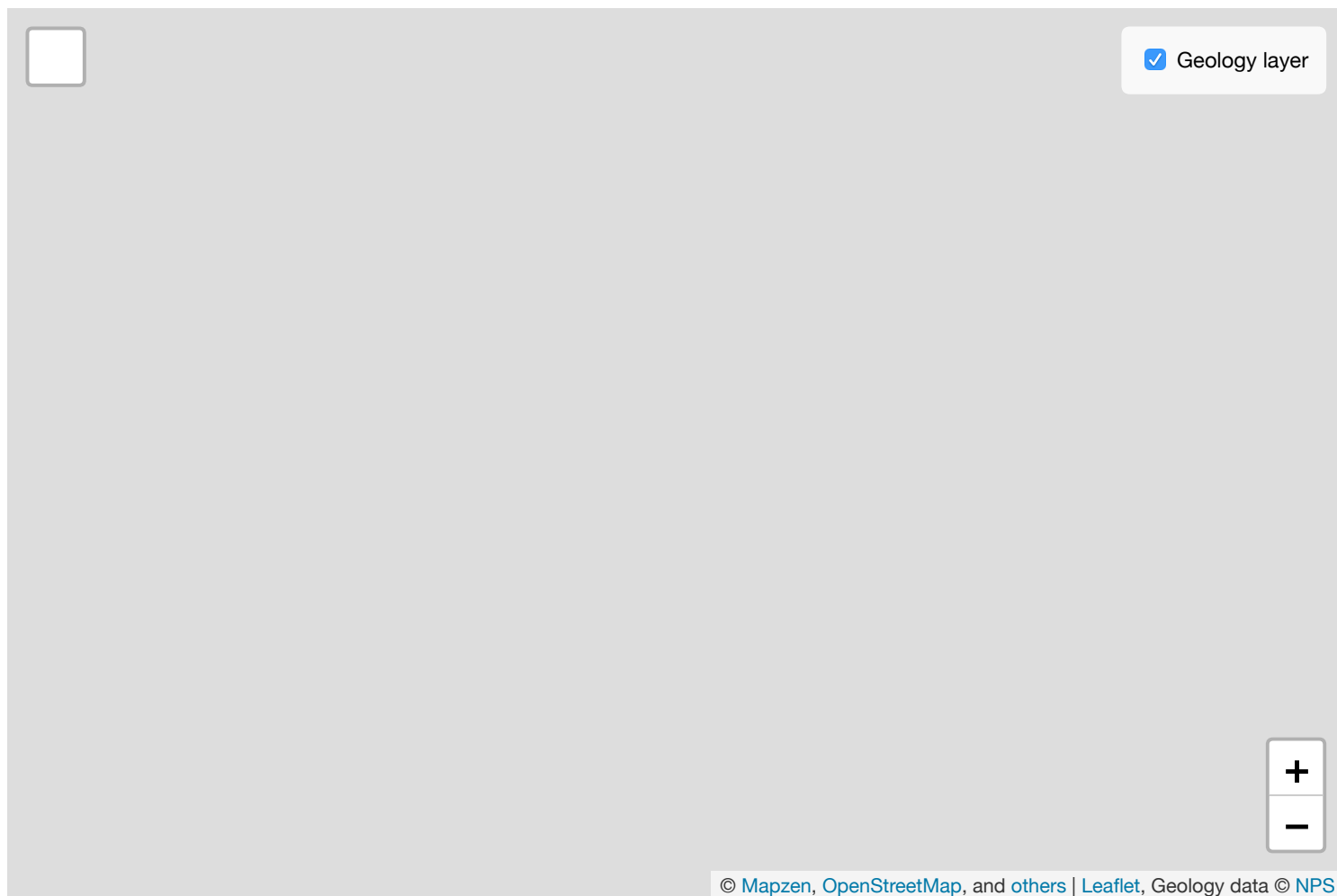
Let's add a function that updates our global variable, which is stored in the `scene.config` object we used earlier. Once again, we'll need to call `scene.updateConfig()` in order to update the scene with our new changes:

```
// ADD HIGHLIGHT  
function highlightUnit(geology_symbol) {  
  scene.config.global._highlight = geology_symbol;  
  scene.updateConfig();  
}
```

Then, we just need to call this function from within our `onMapClick` callback. We already know how to pull out the `GLG_SYM` property from our selected feature, so this part should feel familiar:

```
// ADD SELECTION EVENTS  
function onMapClick(selection) {  
  if (selection.feature) {  
    var latlng = selection.leaflet_event.latlng;  
    var label = selection.feature.properties.GLG_SYM;  
  
    showPopup(latlng, label);  
    highlightUnit(label);  
  } else {  
    highlightUnit(false);  
  }  
}
```

And that's it! Here's the map in all of its interactive glory:



ooooh! clapping! and fanfare!

The final code for this exercise can be found on **bl.ocks.org**
(<https://bl.ocks.org/rfriberg/00eb1b5f98496bf8e3875ee8fa9c9152>).

Thanks for following along with yet another installment of **Make Your Own**
(<https://mapzen.com/tag/make-your-own/>) [[Interactions](#)].

If there are topics you'd like to see covered in a future post, **let us know**
(<https://twitter.com/mapzen>).

And, as always, if you have questions or want to show off something you've made with Mapzen,
drop us a line (<mailto:hello@mapzen.com>). We love to hear from you!

~~~

Check out additional tutorials from the **Make Your Own** (<https://mapzen.com/tag/make-your-own/>) series:

- **One Minute Map** (<https://mapzen.com/blog/one-minute-map/>)
- **Map Sandwich** (<https://mapzen.com/blog/map-sandwich/>)
- **Filters & Functions** (<https://mapzen.com/blog/filters-and-functions/>)
- **Put A Label On It** (<https://mapzen.com/blog/tangram-labels/>)
- **Interactive Mapping with Tangram** (<https://mapzen.com/blog/tangram-interactivity/>)
- **Lots of Dots** (<https://mapzen.com/blog/lots-of-dots/>)
- **Customize Your Search** (<https://mapzen.com/blog/customize-your-search/>)

· 14 December 2016 ·



**Rhonda Friberg**

Rhonda is a javascripter who makes maps, trouble, and the occasional pie.

---

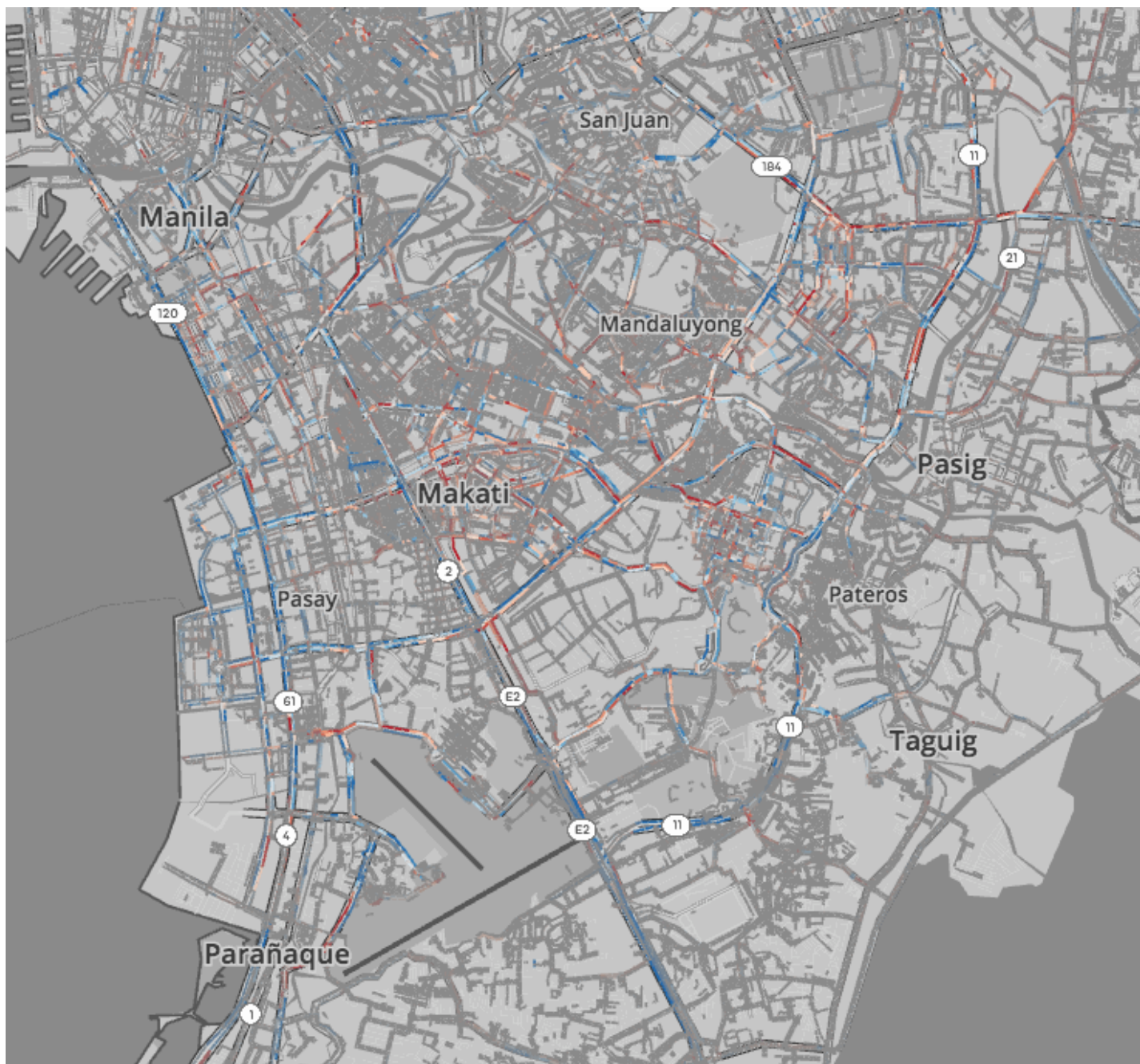
© 2017 Mapzen

# World Bank + Mapzen + partners = open traffic data

mobility (/tag/mobility)

Mapzen is joining **Open Traffic (<http://opentraffic.io>)**, a global initiative led by the World Bank to measure urban traffic congestion. The Open Traffic platform collects anonymous location information from smartphones and converts this raw data into real-time and historical transportation statistics connected to OpenStreetMap. Mapzen is leading development of the platform, and a wide-range of transportation providers are agreeing to contribute anonymized and aggregated data. To learn more about what we'll be doing together, read the World Bank's **press release (<http://www.worldbank.org/en/news/press-release/2016/12/19/the-world-bank-launches-new-open-transport-partnership-to-improve-transportation-through-open-data>)**.

Want to see the potential of Open Traffic? Here's an animated map of traffic in Manila, the most dense city in the world:



(<https://mapzen.github.io/open-traffic-poc-data-demo/>)

( Open as a full-screen map with interactive controls to pan through time ↗

(<https://mapzen.github.io/open-traffic-poc-data-demo/>) )

This based on 11,078,169 measurements of ride-share vehicle positions from **Grab** (<https://www.grab.com>), the largest ride-share operator in Southeast Asia. The positions have been anonymized, aggregated, and summarized to capture the ebb and flow of traffic throughout Manila on August 5, 2016. **Open the map** (<https://mapzen.github.io/open-traffic-poc-data-demo/>) to explore.

Soon these kinds of interactive visualizations and analyses of urban traffic will be simple and accessible for all—no custom data-munging required. We'll also be working with the World Bank and its partners to build traffic-influenced routing into Mapzen Turn-by-Turn. **Previously we've demonstrated (/blog/speed-tiles/)** how Valhalla, the open-source routing engine that powers Mapzen Turn-by-Turn, is ready to take real-time and historic traffic conditions into account—soon that will be possible using fully open-source software and open data.

Traffic is a shared problem. Let's work together on the solution using an open, shared data platform. Read more about it on the **World Bank's feature story** (<http://www.worldbank.org/en/news/feature/2016/12/19/open-traffic-data-to-revolutionize-transport>).

· 19 December 2016 ·



**Alyssa Wright**

Alyssa Wright does community where the phone and meeting are her superpowers of choice.



**Drew Dara-Abrams**

Drew leads Mapzen Mobility products (and aspires to being a flâneur).

---

© 2017 Mapzen

# Open traffic data is the best traffic data

mobility (/tag/mobility)

*"A good science fiction story should be able to predict not the automobile but the traffic jam."* - Frederik Pohl

Mapzen loves transit. So we got excited last week when Elon Musk hinted at a traffic solution through underground tunnels:

Traffic is driving me nuts. Am going to build a tunnel boring machine and just start digging...

— Elon Musk (@elonmusk) **December 17, 2016**

(<https://twitter.com/elonmusk/status/810108760010043392>)

While we'd welcome any new rapid-transit initiatives for inclusion into **Transitland** (<https://transit.land/>), we've been hard at work on a different approach to the same problem: *Open Traffic*. **Announced yesterday by our partners at the World Bank** (<http://www.worldbank.org/en/news/press-release/2016/12/19/the-world-bank-launches-new-open-transport-partnership-to-improve-transportation-through-open-data>), Open Traffic is a data partnership that now includes global ridesharing companies like Easy Taxi, Grab, and Le.Taxi (Have data and want to join? **Email me** (<mailto:randy@mapzen.com>) and I'll connect you to the right folks). Mapzen is building the system to take anonymized real-time data from partners and turn it into usable traffic data for routing, display, and historical analysis.

Why is this important? Mapzen believes that open, collaborative map data is *the best* map data, and traffic data has been too difficult to access openly for too long. Innovative companies like Google, Waze, and Uber have built great products, but we want to make the underlying map data openly accessible so many others can build new products we haven't even imagined yet. Some cities and states **offer free, open datasets** (<https://github.com/graphhopper/open-traffic-collection>) but few of them can achieve the kind of real-time data that app companies can just by turning their user's phones into traffic sensors. Pooling global data offers an accessible solution for everyone, thus the Open Traffic project.



Anonymized, opened, and shared, traffic data will become a valuable public good. The World Bank is involved because traffic is a massive economic problem globally, and open data in this area will improve economies and reduce poverty. We see hints of these benefits in the **test city of Manila** (<https://mapzen.github.io/open-traffic-poc-data-demo/>), but the system we're building now will have global coverage for use everywhere. Last year in New York City, **Uber used data to its advantage** (<https://techcrunch.com/2015/07/22/uber-releases-hourly-ride-numbers-in-new-york-city-to-fight-de-blasio/>): as a popular app, it had access to better data than the local government. With Open Traffic, historic traffic data will become a public benefit, helping transit planners, economies, and citizens everywhere.

Innovations in software engineering combined with traffic and mapping data are poised to transform mobility. Open Traffic will allow us to account for the future of automobiles as well as their future traffic jams. **Our routing engine** (<https://mapzen.com/products/turn-by-turn>) is multi-modal, meaning it provides walking, biking, transit, and driving directions. By understanding better the true costs of traffic we can help communities worldwide decide the best path forward for them.



Image Credits: **Tony Fisher**

(<https://www.flickr.com/photos/tonythemisfit/4099700944/in/photolist-7fh3g3-nEgRD1-8qtMHs-oGHWNH-576Y39-qtKzjY-4nFqth-5sBEfa-5poHPe-HR6xZ2-4wKrnM-qSNfMQ-qKr4g7->



5sBEf6-4uqiWU-6PXeMb-6cw5mc-6pvit1-a1F323-nS22KR-4Aboy1-fN2USJ-dPhBtD-8wnWBa-dkiWp-3k21YQ-5VARin-8njXYF-oPk92E-r2W5Rm-r43Csy-imPRmt-ppNaru-gczK2T-n3WPiV-9Y6vKY-j5xWVs-dpCa9M-4MDqWH-8e6ocE-mJS36M-mQTXQT-q6rdJ8-r35cy2-afmUjB-7aX9F-kZL46G-mQVuC1-dXVdxY-n4cxR)

· 19 December 2016 ·



**Randy Meech**

Billerica Memorial High School graduate. BA, MTS. Mapzen CEO 2013-present.

---

© 2017 Mapzen



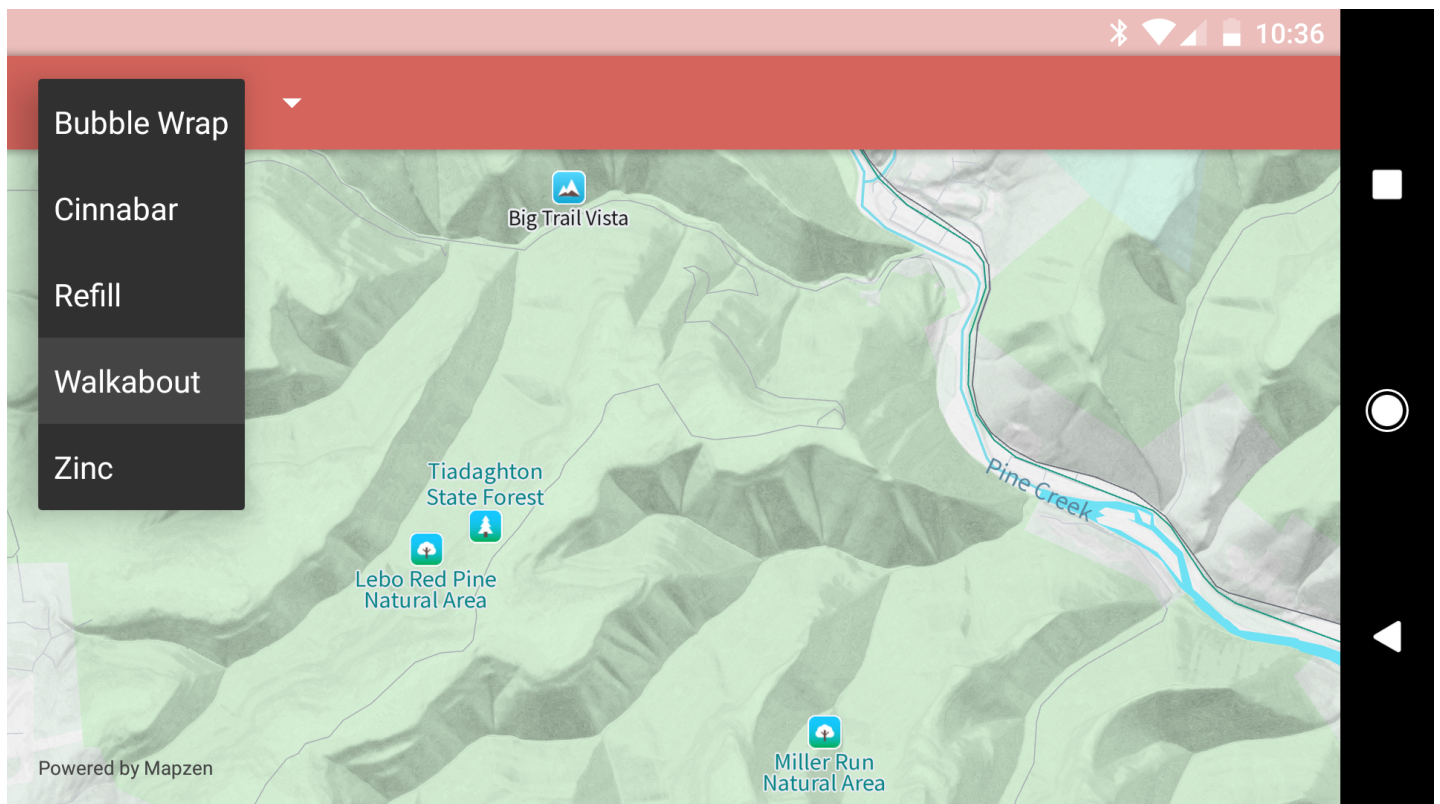
# Mapzen Android SDK 1.2

**mobile** (/tag/mobile) **android** (/tag/android)

Today we are announcing **version 1.2 of the Mapzen Android SDK** (<https://github.com/mapzen/android/releases/tag/v1.2.0>). This release makes significant updates to graphics, routing, and location services. It improves stability and introduces several new features including:

- **Vector tiles 1.0 support** (<https://mapzen.com/blog/v1-vector-tile-service/>)
- **Unified API keys** (<https://mapzen.com/blog/unified-api-keys/>)
- **Multilingual turn-by-turn directions** (<https://mapzen.com/blog/narrative-language/>)
- **Location services improvements** (<https://github.com/mapzen/lost/releases/tag/lost-2.1.2>)

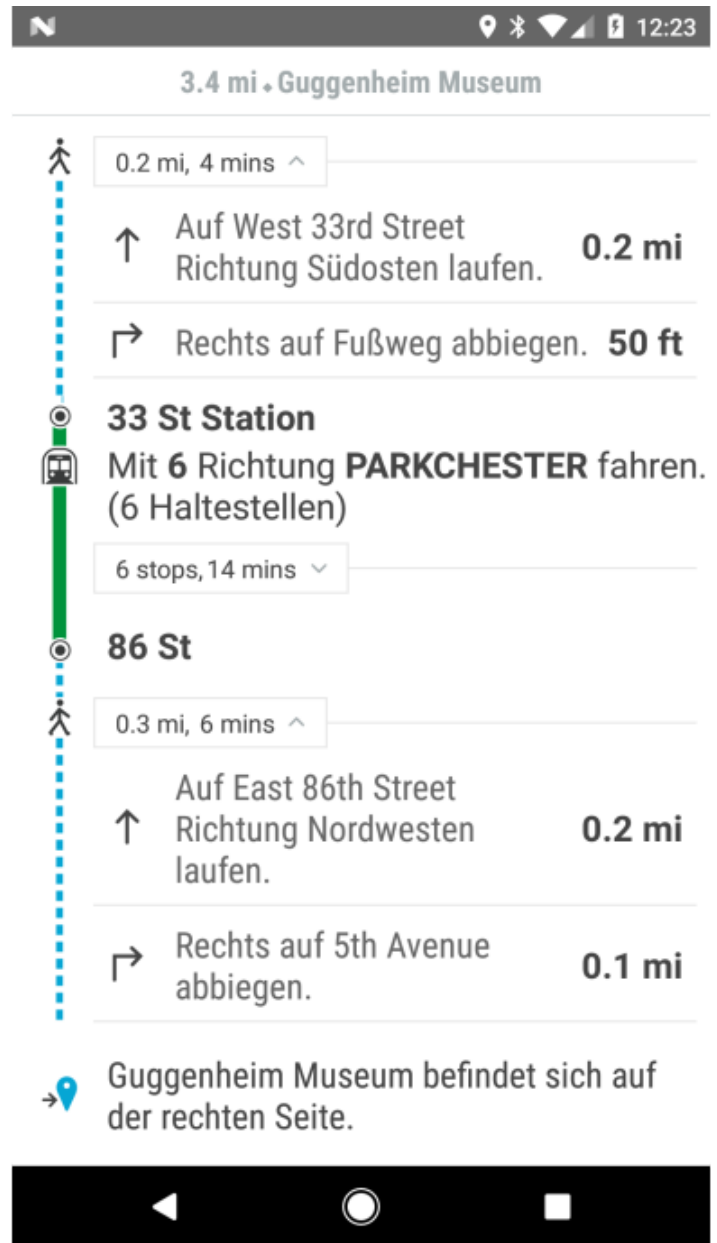
For a full list of changes check out the **release notes** (<https://github.com/mapzen/android/releases/tag/v1.2.0>).



## Style switcher in Mapzen Android SDK demo app

If you are a developer currently using Mapzen on Android we hope you'll give this release a spin and **let us know what you think** (<mailto:android-support@gmail.com>).

If you are a new developer interested in using Mapzen on Android you can find our **installation** (<https://mapzen.com/documentation/android/installation/>) and **getting started** (<https://mapzen.com/documentation/android/getting-started/>) guides here. We would love to hear from you too!



Multimodal directions to Guggenheim Museum ... in German!

Finally if you want to test drive an app built from the bottom up using Mapzen tools and services you can check out the latest release of our reference application **Eraser Map available in the Google Play store** (<https://play.google.com/store/apps/details?id=com.mapzen.erasermap>).

· 22 December 2016 ·



### Chuck Greb

Chuck is an open source enthusiast, test-driven evangelist, and clean code connoisseur of all things mobile.

---

© 2017 Mapzen

